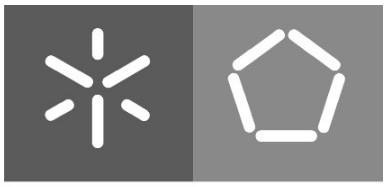

Universidade do Minho
Departamento de Sistemas de Informação

Nuno Alexandre de Albuquerque Costa

Planeamento e Gestão Concorrente das Equipas *Scrum*: Das Arquiteturas Lógicas aos Métodos Ágeis

Trabalho orientado por:

Professor Doutor Ricardo J. Machado

Mestrado em Sistemas de Informação

Outubro de 2013

DECLARAÇÃO

É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA DISSERTAÇÃO/TRABALHO PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

Universidade do Minho, 31 de Outubro de 2013

Assinatura: _____

Agradecimentos

À Filipa, porque foi devido ao seu apoio e persistência que o trabalho chegou ao fim;

Aos meus pais, pelo esforço que toda a vida fizeram para me proporcionarem os melhores valores e a melhor educação possível;

À minha família, com especial destaque para a Nini e para o Diogo, por sempre me terem dado o apoio que necessitei;

Aos meus colegas, com especial destaque para o Pedro Oliveira e para o José Arnaud, pelos sábios ensinamentos e pela amizade que ultrapassou idades e distâncias;

Aos meus colegas de trabalho, nomeadamente ao Carlos Oliveira, pela motivação, ajuda e companhia no dia a dia;

Ao meu orientador, Professor Dr. Ricardo Machado, por ter aceite um orientando numa situação tão especial;

Ao Professor Filipe de Sá-Soares, por ter sido uma voz inspiradora durante todo o mestrado e o Professor que mais me marcou na minha carreira académica;

Aos meus amigos, porque um Homem, sem grandes amigos, quaisquer que sejam, estará perdido.

Resumo

Ao ritmo que as organizações necessitam de novos produtos, novas tecnologias, e novas abordagens, é necessário que a indústria das empresas de Tecnologias de Informação esteja apta a responder de forma célere a todas as solicitações. Os requisitos que hoje são considerados essenciais para um produto, podem ser amanhã dispensáveis e as tecnologias exigidas ontem, com certeza que estarão obsoletas no final do dia de hoje. A inflexibilidade não é uma opção e os produtores de *software* devem adoptar medidas que abracem a mudança constante de requisitos e tecnologias como uma oportunidade de alavancar a qualidade dos seus produtos.

As novas abordagens para a gestão de projectos de *software* são as chamadas metodologias ágeis, estruturadas em ciclos de desenvolvimento curtos, permanentemente avaliados pelo cliente, onde a comunicação assume um papel fundamental para o alinhamento de expectativas entre as partes. Apesar desse menor rigor e formalismo, existem projectos que ainda necessitam de modelações formais e complexas para dar suporte à implementação do produto, garantindo que todos os potenciais pontos críticos e de comunicação foram identificados à partida.

Neste trabalho vão-se conjugar as metodologias ágeis, nomeadamente o *Scrum*, com a gestão de requisitos e modelação de sistemas provenientes de abordagens mais formais. Para o efeito, foi formulado um processo que produzirá um conjunto de User Stories partindo de um diagrama UML de componentes, criado a partir da aplicação do 4SRS (*4-Step Rule Set*). Adicionalmente, o modelo criado será aplicado num caso prático, assim como serão efectuadas recomendações para adaptar os procedimentos das equipas *Scrum* a trabalhar em paralelo no referido projecto, procurando que as mesmas implementem o melhor produto possível no mais curto espaço de tempo.

Abstract

At the pace that today's organizations require new products, new technologies, and new approaches, it is necessary for the industry of Information Technology companies to respond swiftly to every demand. Requirements that are essential today may become completely expendable tomorrow, just like the technologies that were appropriate yesterday will be completely obsolete by this afternoon. Inflexibility is not an option nowadays, so software producers must include new procedures that embrace change as an opportunity to leverage the quality of their products.

New approaches to Project Management come from the so-called agile methodologies, which are supported by small development cycles, continually assessed by the client and where communication takes a lead role in aligning expectations among stakeholders. Despite this lower degree of formalism, there are still projects that need formal modelling and documentation in order to raise and handle critical issues from the very beginning of the project.

This work aims to bridge agile methodologies (namely Scrum) with requirements management and architecture modelling procedures from older Project Management methodologies. For this purpose, a process was formulated that takes an UML component diagram as input (generated by the application of 4-Step Rule Set) and generates a list of *User Stories,* consistent with development teams that follow agile approaches.

Finally, this process was tested in a practical project and recommendations were established to tailor *Scrum* procedures to the teams working in parallel in said project, with a clear goal in mind: creating the best possible product in the shortest amount of time.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

De modo a simplificar a leitura regular de certas expressões, optou-se por redigir uma lista de acrónimos, nomeadamente:

- 4SRS – *4 Step Rule Set*
- AE – *Architectural Element*
- CMMI - *Capability Maturity Model Integration*
- FLAUS – *From Logical Architectures to User Stories*
- RUP – *Rational Unified Process*
- UML – *Unified Modeling Language*
- XP – *eXtremme Programming*

Planeamento e Gestão Concorrente das Equipas *Scrum*: Das Arquitecturas Lógicas aos Métodos Ágeis*

* este trabalho foi escrito ao abrigo do antigo acordo ortográfico.

1. Introdução

1.1 Contextualização

Segundo os dicionários da Oxford[1], gerir (ou em inglês *manage*) tem, entre outros significados, o de "ter êxito em sobreviver ou em conseguir atingir o objectivo, não obstante as circunstâncias difíceis". A arte de gerir o esforço disponível é experimentada desde o período da idade média, onde os Egípcios geriam o esforço e coordenavam os seus escravos para conseguirem a construção das pirâmides de uma maneira mais eficiente. Para outros autores, o acto de gerir é visualizado como um tipo de filosofia, onde se considera a gestão como "*the art of getting things done through people*" (em português traduzido para "a arte de fazer as coisas através das pessoas")[2].

A gestão é, portanto, uma área de estudo que concentra mentes e ideias desde há séculos, onde existem várias correntes e perspectivas que têm vantagens e desvantagens, que têm fieis seguidores e ferozes detractores. No contexto do desenvolvimento de *software*, a mesma rotura de perspectivas persiste. Se durante anos foi utilizado o modelo cascata – seguindo as influências de Henry Ford e da sua linha de fabrico em série - onde a produção de software seguia um processo de faseamento bem definido, actualmente já existem bastantes académicos e profissionais do sector que defendem a agilização do processo que guia a produção de software.

Numa área onde o emergir constante de novas tecnologias e novos processos pode desactualizar produtos e tornar obsoletos certos requisitos num curto espaço de

[1] Dicionários da Oxford disponíveis para consulta online em: http://oxforddictionaries.com

[2] Esta frase não tem uma citação associada devido ao facto de não se conseguir verificar, com toda a certeza, quem a proferiu. No entanto, as evidências existentes apontam para Mary Parker Follet, autora estado-unidense que escreveu sobre temas relacionados com a gestão e os recursos humanos.

tempo, as organizações sentem a necessidade de ter equipas adaptáveis a qualquer alteração de contexto. Aliado a este "jogo de cintura", as empresas de *software* não podem descurar a qualidade, enquanto mantêm a eficiência de utilização de recursos, de modo a poderem fazer frente aos grandes produtores que despontam de países como a Índia ou a China, que possuindo recursos humanos tecnicamente capazes, apenas recentemente começaram a apostar na solidificação e maturidade dos seus processos.

Foi a meados do séc. XX, com o aparecimento dos primeiros grandes projectos de criação de *software,* que foi sentida a necessidade de criar um processo transparente, paralelo a todas as organizações, que guiasse na criação de novos produtos de *software.* Fases definidas, documentação e boas práticas foram redigidas e revistas diversas vezes ao longo dos anos, originando uma metodologia denominada de modelo *waterfall* (ou cascata), em que o *software* deveria ser desenvolvido em fases específicas (planeamento, análise, *design*, implementação, manutenção) e bem demarcadas umas das outras [Boehm, 2002; Yang et al., 2008].

Com a massificação e o aumento do alcance das tecnologias e com a abertura das organizações mais pequenas às novas tecnologias, assistiu-se a uma procura acentuada por produtos de *software*, muitos deles adaptados à realidade de cada organização. Novas tecnologias, requisitos constantemente alterados e diferenças de expectativas e perspectivas eram factores que recorrentemente afectavam as relações entre produtores e clientes. Com o modelo cascata, os clientes não tinham grande envolvimento com o processo de criação dos produtos, apenas participando na fase de elicitação de requisitos e na fase de avaliação e instalação do produto. Foram sucessivos os casos de clientes que recebiam produtos que, afinal, não iam de encontro às suas necessidades de negócio ou estavam desadequados dos seus pedidos, após terem sido gastos avultados recursos económicos na sua criação. Assim, não era de admirar que devido a derrapagens orçamentais, temporais ou a requisitos não implementados, em 1994 existissem apenas 16% de projectos que eram considerados um sucesso, sem desvios de maior.

	1994	1998	2002	2006	2009
Sucesso	16%	26%	34%	35%	32%
Alterado	53%	46%	51%	46%	44%
Cancelado	31%	28%	15%	19%	24%

Tabela 1.1 – Dados referentes ao sucesso dos projectos de *software*, recolhidos pela Standish Group [Dominguez, 2009]

Apesar de algumas melhorias verificadas com o passar dos anos, em 2009 eram ainda quase um quarto dos projectos existentes que seriam cancelados devido a desvios orçamentais, de prazos ou de requisitos não implementados. Assim, no final do séc. XX, assistiu-se ao emergir de novas metodologias e experiências que tentassem contrariar a inflexibilidade e a falta de resposta a mudanças no ambiente. A escassez de maleabilidade para se adaptar a novos contextos foi visto por vários autores como uma das principais causas para o cancelamento de vários projectos [Cho, 2008] .

Em 1984, Hirotaka Takeuchi e Ikujiru Nonaka, dois professores universitários, revisitaram a hipótese de flexibilizar os processos de criação de produtos de *software*, apostando num método holístico sustentado em várias características disruptivas com o panorama de então, apoiadas em exemplos de grandes organizações que já as colocavam em prática, como são exemplos a Honda, a Fuji-Xerox ou a Epson [Takeuchi & Nonaka, 1986]. Características como equipas geridas autónomamente ou fases de desenvolvimento sobrepostas eram as novidades, que mais tarde serviriam de inspiração para os defensores e criadores de algumas metodologias que hoje chamamos de ágeis. Ao contrário do natural ciclo de vida das metodologias de gestão, que emergem de estudos e hipóteses concretizadas pelo meio académico, estas novas ideias despontaram naturalmente no seio das organizações, devido à vontade de melhorar o seu processo, tornando-se mais eficientes, mais resistentes às volatilidades tecnológicas e imunes à instabilidade natural do próprio conceito de criar *software* novo. Assim, nasceu a perspectiva ágil, baseada na ideia que cada produto é único, potencialmente ambíguo e naturalmente complexo, ao invés da demanda por planeamento, controlo e rigidez naturais das metodologias tradicionais de desenvolvimento de *software* [Moe et al., 2010]. O *Scrum*, juntamente com o *eXtreme Programming* (XP), são duas das abordagens metodológicas que nasceram na procura pela flexibilidade, regidas pelos princípios do Manifesto Ágil[3]. Hoje em dia, são amplamente reconhecidas na indústria de produção de software como mecanismos que ajudam na produtividade e melhoria da qualidade do software, com casos práticos a darem suporte a esta conclusão [Bates & Yates, 2008; Moore et al., 2007; Schatz & Abdelshafi, 2005; Sutherland & Altman, 2009; Sutherland & Johnson, 2008].

[3] Declaração de princípios que fundamentam o desenvolvimento ágil de software, disponível online em http://agilemanifesto.org e no Anexo I.

1.2 Objectivos

Suportado pelo aumento da popularidade e da utilização do *Scrum*, o principal objectivo deste trabalho é averiguar a pertinência da criação de um mecanismo que consiga gerar requisitos compatíveis com as equipas que seguem esta abordagem metodológica, a partir da arquitectura já definida para um produto. A utilização do método 4SRS (*four step rule set*), com capacidade para gerar arquitecturas lógicas a partir de diagramas de Casos de Uso, já foi utilizado com sucesso no panorama nacional de desenvolvimento de *software*, continuando a ser alvo de estudos e melhorias na comunidade académica.

Propõe-se então neste trabalho a criação de um processo, compatível e complementar ao 4SRS, que consiga gerar requisitos compatíveis com *Scrum* (apelidados de *User Stories)*, facilitando assim a tarefa de especificação de requisitos, nem sempre simples de executar, e para o qual as empresas mais pequenas quase nunca estão devidamente preparadas.

O primeiro objectivo consiste numa análise cuidada do *Scrum* em todas as suas vertentes, identificando as suas práticas comuns, os seus principais benefícios e limitações. Como a sua criação e utilização ocorre maioritariamente no contexto prático e empresarial, a recolha de informação que caracteriza o *Scrum* foi efectuada utilizando vários artigos científicos, casos práticos, *blogs*, comunidades e fóruns *online*, e da própria experiência do autor do trabalho como integrante de uma destas equipas.

Conhecido em detalhe o *Scrum*, o segundo objectivo consiste na criação de um processo capaz de utilizar o resultado final da aplicação do 4SRS e gerar um conjunto de *User Stories* que representem o conjunto total das funcionalidades do sistema a implementar.

O terceiro objectivo envolve a aplicação prática dos dois primeiros num caso real. O projecto ISOFIN, no qual a Universidade do Minho participa efectuando a especificação, perfilava-se como uma boa hipótese por ter sido aplicado o processo de 4SRS para gerar a sua arquitectura lógica. Utilizando o processo criado para responder ao segundo objectivo, foram geradas as *User Stories* que mapeiam os requisitos pretendidos para o produto. Adicionalmente, foi sugerida uma configuração para as equipas de implementação, com base no *Scrum*, capazes de pegar nas *User Stories* geradas e partir directamente para a sua implementação.

1.3 Estrutura do Documento

De modo a atingir os objectivos especificados na secção anterior, este trabalho divide-se em cinco capítulos distintos, que interligados entre si permitem responder ao principal objectivo que esta dissertação pretende concretizar.

No primeiro capítulo é dado um enquadramento breve ao panorama da gestão de projectos de *software* nos últimos anos, com uma ligeira introdução à mudança de paradigma vivenciada.

O segundo capítulo consiste em detalhar minuciosamente as novas práticas na gestão de projectos que se baseiam em metodologias ágeis. É apresentado o *Scrum*, listando o tipo de actores existentes, os eventos e os artefactos que estão presentes no dia a dia das equipas. É também descrito a maneira como os requisitos funcionais dos produtos são tratados no âmbito do *Scrum – User Stories*.

No terceiro capítulo é exposto o processo FLAUS (*From Logical Architecures to User Stories)*, processo que gera *User Stories* a partir do resultado da aplicação do 4SRS.

Ao longo do quarto capítulo o processo FLAUS é aplicado na prática, gerando as *User Stories* para o projecto ISOFIN. Paralelamente, são apresentados algumas sugestões de configuração das equipas que têm responsabilidade de implementar o produto.

No quinto capítulo, são resumidos de forma breve os esforços evidenciados na prossecução dos objectivos definidos, com considerações e conclusões relativas ao trabalho efectuado, apresentando as suas limitações e sugerindo melhorias e tarefas futuras capazes de enriquecer o processo criado no âmbito desta dissertação.

2. O *Scrum* e as suas Utilizações

2.1 Introdução aos Métodos Ágeis

Durante as últimas décadas, o modelo de cascata foi responsável por balizar a grande maioria dos projectos de *software*, com os resultados conhecidos pela Tabela 1.1. Os factores que influenciavam a sua utilização prendiam-se com o facto de serem modelos que, na teoria, conseguiam prever com bastante detalhe quais os recursos (temporais, económicos ou humanos) que seriam necessários para terminar um projecto com sucesso. Estes métodos tradicionais de desenvolvimento de *software,* pesados e sujeitos a extensos planeamentos, dependiam de requisitos especificados e documentados no início, que não mudassem durante o desenvolvimento do produto. Por isso, era suposto que os clientes soubessem o que realmente necessitavam antes de ver o produto final e que o processo de desenvolvimento de *software* fosse totalmente previsível e repetível [Sutherland, 2001].

No entanto, seguindo os princípios da incerteza dos requisitos de Humphrey, que preconiza que num novo sistema os requisitos não são completamente conhecidos até ao cliente experimentar realmente o produto, aliado aos princípios de Ziv [Sutherland & Schwaber, 2010; Ziv & Richardson, 1997] que afirma que a incerteza é um dado adquirido e inevitável nos processos de desenvolvimento de produtos de *software,* não espanta a taxa de cancelamento de projectos de quase 31% em 1995, onde as principais causas de cancelamento prendiam-se com levantamento incompleto de requisitos e falta de envolvimento dos clientes durante a implementação do produto [Boehm, 2000].

As fases iniciais de levantamento e documentação de requisitos revestiam-se de carácter fundamental, uma vez que estes influenciavam integralmente todas as fases futuras e respectivos planeamentos. Na realidade, a verdade é que frequentemente os clientes desconheciam o que era realmente necessário para o seu negócio, originando variadas mudanças de requisitos durante a vida útil do projecto. Isso causava bastantes retrocessos e avanços, com as suas implicações orçamentais e temporais, devido à pouca flexibilização dos modelos tradicionais de desenvolvimento de *software*, assim como à distância muitas vezes existente entre o cliente e o fornecedor do produto. No caso de existir uma excelente ideia para um projecto, já tendo sido ultrapassada a fase inicial de planeamento, essa ideia era vista como uma potencial ameaça ao projecto, ao invés de ser olhada como um potencial factor de sucesso [Deemer et al., 2010].

No final do séc. XX, começaram a surgir alguns esforços, um pouco por todo o mundo, para mudar o paradigma no qual estavam assentes os projectos de produção de *software*. Ao invés de contrariar e rejeitar a mudança, foram experimentadas alternativas para aceitar a mudança, integrando-a como uma inevitabilidade e um potencial factor para aumentar a taxa de sucesso dos produtos. A constante comunicação e negociação com o cliente deixou de ser vista como um factor originário de derrapagens financeiras e temporais, passando a ser visto como uma oportunidade para melhorar o produto final utilizando os mesmos recursos, aumentando assim a eficiência na criação de produtos de *software.*

Para trocar experiências e solidificar este novo rumo, um grupo de 17 pessoas ligadas à indústria do *software* reuniram-se em Fevereiro de 2001 em Utah, Estados Unidos, para discutirem os chamados “processos leves”. Como consequência desse encontro foi redigido o Manifesto Ágil, documento onde redundam 12 princípios (ver Anexo I – Os Doze Princípios do Manifesto Ágil) que guiam a produção de *software* de um modo ágil, baseados nos seguintes valores fundamentais [Fowler, 2001]:

- Indivíduos e interacções mais do que processos e ferramentas;
- Software funcional mais do que documentação abrangente;
- Colaboração com o cliente mais do que negociação contratual;
- Responder à mudança mais do que seguir um plano.

Todas estas ideias representam disrupções significativas com os princípios seguidos nos modelos tradicionais de desenvolvimento de *software*. A comunicação é elevada a um patamar de enorme importância, ainda que à custa de uma menor

maturidade nos processos; o *software* tem como principal objectivo a sua funcionalidade, mesmo que não seja suportado por uma documentação extensa; a colaboração com o cliente é primordial e tem de ser constante, de maneira a que este se mantenha sempre a par dos desenvolvimentos, com a possibilidade de introduzir novas ideias e conceitos que possam alterar o produto, inclusivé durante a sua fase de desenvolvimento; existe abertura a alterar contextos, mesmo que isso tenha como consequência planificações obsoletas.

Outra grande diferença experimentada pelos métodos ágeis consiste na divisão do trabalho em pequenas iterações, que vão sendo constantemente avaliadas e corrigidas pelos intervenientes no processo. Pelo contrário, as abordagens mais tradicionais compreendem a entrega de um produto em poucas fases, com poucos momentos de avaliação por parte dos clientes. Esta iteratividade permite optimizar a previsibilidade (que é diminuta nos métodos ágeis comparativamente aos métodos tradicionais) e aumentar o controlo do risco, dado que os potenciais problemas do produto são descobertos numa fase inicial da implementação [Rockwood, 2007].

Guiados por esta nova matriz de pensamento, brotaram diversas experiências e casos práticos que levaram ao aparecimento de algumas metodologias ágeis de desenvolvimento de *software*: AUP (*Agile Unified Process)*, *Crystal Methodologies*, DSDM (*Dynamic Systems Development Model*), XP (e*Xtremme Programming)*, FDD (*Feature Driven Development*), OpenUP (*Open Unified Process)* ou *Scrum* são alguns dos nomes das principais técnicas emergentes. Destas várias técnicas, destacam-se o *Scrum* e o XP como as técnicas que mais aceitação tiveram na indústria de produção de *software*, graças à sua fácil compreensão e ao facto de contrariarem precisamente as principais causas para o cancelamento dos projectos de *software* (requisitos incompletos no início do projecto e distanciamento para com o cliente).

2.2 O *Scrum*

Origem. Intrigados com as técnicas expostas por Takeuchi e Nonaka em 1984 e guiados pela necessidade de flexibilizar o processo de desenvolvimento para abraçar a mudança e a incerteza, Ken Schwaber e Jeff Sutherland formalizaram uma nova abordagem metodológica ágil chamada *Scrum*. Foi em 1993, na Easel Corporation, que os autores aplicaram o *Scrum* pela primeira vez, tendo sido formalizado em 1995 na OOPSLA'95 [Schwaber, 1997; Sutherland, 2004].

A palavra *Scrum* tem origem no desporto colectivo *rugby*, que representa a colocação da bola em jogo e onde toda a equipa reúne esforços para conquistar o objectivo: a posse de bola. Fazendo o paralelismo com a área de desenvolvimento de *software*, o *Scrum* depende, entre muitas outras coisas, de um trabalho de equipa constante e de um compromisso, disponibilidade e entrega total para conseguir ter sucesso e atingir os objectivos.

Transparência, Inspecção, Adaptação. Schwaber e Sutherland são autores de diversos artigos que explicam as motivações que originaram o *Scrum*. Como foram ambos participantes activos no encontro que originou o manifesto ágil, é natural que a abordagem metodológica do *Scrum* seja conciliável com todos os princípios presentes no documento. Com a principal força motriz a ser o trabalho em equipa, a comunicação e a abertura à mudança e incerteza, o *Scrum* é uma metodologia baseada num processo empírico e é norteada por três aspectos base: transparência, inspecção e adaptação, explicados em detalhe no Guia do *Scrum* [Schwaber & Sutherland, 2011].

A transparência é um adjectivo comum e fundamental nos projectos regidos pelo *Scrum*. Os autores consideram que, ao contrário do que acontece com os métodos tradicionais de desenvolvimento de *software*, os aspectos significativos do processo devem ser visíveis para todos os que são responsáveis pelos resultados, incluindo todos os *stakeholders* que no *Scrum* têm um papel preponderante. Apenas utilizando uma linguagem comum, entendida por todos os intervenientes, se consegue a união necessária para conseguir atingir os objectivos.

A inspecção é outro conceito importante para o sucesso dos projectos. Os artefactos utilizados pelo *Scrum* e o progresso face ao objectivo final devem ser constantemente monitorizados, de modo a descortinar eventuais desvios em relação ao programado. Estas inspecções não devem afectar a execução normal das tarefas e devem ser realizadas por indivíduos qualificados de modo a serem mais eficazes.

Os momentos de inspecções periódicas – feitos pelas equipas num momento chamado *Sprint Retrospective* - indicam que a própria abordagem metodológica do *Scrum* não é estanque na vida de um projecto, podendo os seus eventos, actores e artefactos serem modificados a meio, se com essa acção resultarem melhorias para a equipa e/ou produto. Destas melhorias provém a adaptação. Os ajustes efectuados para aperfeiçoar o processo devem ser realizados assim que as inspecções encontrarem

algum problema, de modo a minimizar os desvios tão cedo quanto possível. Este ciclo interminável de inspecção – adaptação tem nuances únicas em cada projecto, pelo que as acções e medidas a tomar terão de ser pensadas para o projecto em questão, não existindo nenhum mandamento único capaz de resolver todos os problemas encontrados. O *Scrum* é assim uma metodologia ágil suportada pela Teoria de Controle Empírica [Cho, 2008; Schwaber, 1997; Sutherland & Altman, 2009].

Autonomia. Outra das principais características das equipas geridas pelo *Scrum* é o elevado grau de autonomia presente. A falta de uma figura decisória e rígida como o gestor de projecto, que responde a uma só voz pelo sucesso ou insucesso da equipa, faz com que a equipa se sinta mais responsável pelo resultado do seu trabalho. Aliás, a autonomia e gestão autónoma é uma característica fundamental para o êxito destas equipas [Moe et al., 2008]. Esta confere um sentimento de transcendência e compromisso aos elementos, dado que o planeamento (seja as horas necessárias para realizar a tarefa assim como as próprias tarefas realizadas) é totalmente decidido por eles, e assim sentem-se mais motivados a atingir os objectivos com que se comprometeram. Além do mais, o *Scrum*, tal como os métodos ágeis, encoraja a partilha de experiências e conhecimentos pelos membros da equipa, procurando que todos saibam o estado do projecto e que, idealmente, qualquer elemento esteja apto para executar qualquer tipo de acção sobre o produto [Sutherland, 2005]. Esta visão é um pouco contra a especialização defendida por outras metodologias como o *Rational Unified Process* (RUP[4]), onde a especialização dos indivíduos é valorizada.

Apesar da autonomia colectiva ser um ponto fundamental, um dos segredos do *Scrum* reside no trabalho de equipa entre os indivíduos que a compõem. É frequente o pensamento de pessoas fora da área da produção de *software* que, para programar as aplicações utilizadas diariamente nos *smartphones*, nos computadores ou em outros dispositivos, é necessário haver indivíduos de carácter anti-social, que trabalham na arte da criação de *software* de maneira isolada, sem qualquer contacto com o mundo exterior. Se é certo que poderão existir casos de profissionais com estas características, a verdade é que a programação na maioria das vezes deve ser vista como uma actividade social, que necessita de trabalho de equipa para ser bem sucedida [Bates &

[4] Processo, da área da Engenharia do Software, que utiliza o paradigma orientado a objetos, juntamente com diagramas UML para ilustrar os processos em acção. O referencial RUP encontra-se disponível em: www.wthreex.com/rup/

Yates, 2008]. Este conceito defendido pela dupla Bates e Yates é levado ao extremo no *Scrum*, onde a partilha de conhecimento e a ajuda mútua são valores decisivos para o sucesso dos projectos.

Para ajudar nesta união entre os intervenientes, é recomendada a disposição de equipas em *open-space*, sem barreiras físicas que delimitem os espaços de cada um e que inibam o contacto entre os actores. Em certos casos práticos estudados, a criação deste ambiente foi um factor fundamental para o sucesso da implementação do *Scrum* e na melhoria dos produtos, dado que aumenta a coesão e a colaboração entre os integrantes da equipa [Moore et al., 2007]. Também a gestão de topo das empresas se deve comprometer com as equipas *Scrum*, dando-lhes apoio e espaço de decisão, atribuindo um voto de confiança que deve ser reconfirmado no final de cada ciclo iterativo bem sucedido [Beavers, 2007].

Casos Práticos. O *Scrum* não nasceu em laboratório. Ao contrário de algumas teorias e mecanismos que controlam os processos de produção das indústrias de qualquer área, que normalmente passam do plano teórico académico para as experiências na prática, o *Scrum* emergiu na própria indústria, tentando combater as causas que propiciavam uma grande taxa de cancelamento dos processos. É portanto natural que na literatura da área apareçam vários casos de estudo reais onde o *Scrum* foi implementado, apresentando a sua estrutura, as vantagens e as dificuldades na implementação.

Pelo mundo fora, são incontáveis as organizações que já adoptaram o *Scrum* no seu processo de desenvolvimento de *software.* Segundo os dados do último inquérito a indivíduos que trabalham na indústria de desenvolvimento de *software*, 80% dos inquiridos afirma que a organização à qual pertencem já utiliza ou utilizou alguma prática condizente com algum dos princípios ágeis de desenvolvimento de *software.* Além disso, dos projectos que são conduzidos por metodologias ágeis, mais de metade (52%) utilizam a abordagem metodológica do *Scrum* [VersionOne, 2011]. Não admira portanto que grandes organizações como a Microsoft, 3M, Oracle, Primavera, Yahoo, Motorola, IBM, Philips, Siemens, HP, entre outras, possuam projectos que utilizam *Scrum*.

Apesar destas gigantes tecnológicas, a utilização do *Scrum* não se esgota no contexto tecnológico. Já foi provado que esta abordagem pode ser utilizada com sucesso em qualquer área de produção, como por exemplo em equipas de *marketing* e

responsáveis pela gestão de produtos [Vlaanderen et al., 2011]. Até mesmo no meio educacional, integrada em disciplinas universitárias de programação, se constata que o *Scrum* pode trazer ganhos de produtividade e na qualidade dos trabalhos [Scharff & Verma, 2010].

Produtividade. A generalidade dos casos estudados demonstram que a produtividade e a qualidade dos produtos aumentou significativamente. Seja pelo aumento da comunicação e do trabalho em equipa, seja pela maior proximidade com o cliente, a verdade é que o paradigma ágil do *Scrum* contribuiu para melhores resultados nos projectos, mesmo aqueles que migraram de outras metodologias mais tradicionais. De modo a perceber até que ponto as organizações estavam a sentir melhorias de produtividade, um conjunto de investigadores do Centro de Informática da Universidade Federal de Pernambuco (Recife, Brasil) conduziu, em 2010, uma análise sistemática de literatura sobre o impacto da implementação do *Scrum* na produtividade das equipas e na qualidade dos produtos. Dos 28 casos analisados, todos reportaram ganhos na produtividade, sempre que esta foi analisada nos respectivos artigos. Além da produtividade, também a satisfação do utilizador, a qualidade do produto, a motivação da equipa e a redução de custos foram observados, com os resultados a serem dispostos na Tabela 2.1 [Cardozo et al., 2010].

Objectivo	Impacto	Nº Artigos
Produtividade	Muito Bom	5
	Bom	7
	Médio	2
Satisfação Cliente	Muito Bom	3
	Bom	2
Qualidade	Muito Bom	2
	Bom	4
Motivação da Equipa	Muito Bom	1
	Bom	4
Redução de Custos	Muito Bom	1
	Bom	2

Tabela 2.1 – Resultados da análise sistemática da literatura sobre a influência do *Scrum* na procura de certos objectivos. Fonte: [Cardozo et al., 2010]

Como se pode verificar pelos resultados da tabela acima, muitos reportaram algumas melhorias na produtividade, sendo que cinco dos casos reportaram até

significativas melhorias. Também nos outros objectivos, todos os casos indicaram grandes melhorias, sendo assinalável que nenhum dos casos estudados referir que o *Scrum* fosse um contratempo para qualquer um dos objectivos. Um dos muitos casos disponíveis em artigos na literatura da especialidade é o exemplo da 3M, empresa tecnológica presente no *ranking* Fortune 500[5]. Nesta organização onde o *Scrum* foi implementado, existiu um consenso alargado sobre a perspectiva que a adopção do *Scrum* tornou o processo de produção de *software* muito mais produtivo, comparativamente com a sua abordagem anterior de modelo em cascata [Moore et al., 2007].

Outros estudos indicam detalhes mais específicos sobre o aumento da qualidade dos produtos de *software* produzidos em equipas sob a abordagem metodológica do *Scrum*. Uma das características de controlo de qualidade de *software* é a quantidade de *bugs,* ou defeitos, encontrados no código. Jingyue Li, juntamente com Nils Moe e Tore Dybå, três autores experimentados na escrita de artigos sobre *Scrum*, conduziram um estudo sobre o impacto do *Scrum* na qualidade do *software*, nomeadamente no número de defeitos reportados e na taxa de correcção desses mesmos defeitos [Li et al., 2010]. Os seus resultados indicam que no âmbito de uma comparação entre um projecto que era anteriormente gerido através de uma metodologia tradicional de desenvolvimento de *software* e o *Scrum*, esta última abordagem resultava em defeitos a serem encontrados muito mais rapidamente (devido à iteratividade do *Scrum*). Isso significa uma resolução mais rápida do problema porque o código tinha sido escrito à menos tempo, representando um menor custo na resolução do problema. Além disso, o facto de não ter existido tempo para construir outras funcionalidades que dependessem do código defeituoso, impede a propagação do erro para outras componentes do produto.

Integração. Embora o *Scrum* seja uma abordagem metodológica com princípios muito próprios, é uma *framework* de procedimentos que pode ser integrada com outro tipo de metodologias ou técnicas. O *Scrum* dá indicações específicas em relação ao processo de gestão da equipa e do produto, mas não apresenta indicações de como efectivamente implementar código. Além da possível integração com outros métodos ágeis (nomeadamente o XP) com excelentes resultados [Kniberg, 2007], é falso que o

[5] Ranking publicado pela revista FORTUNE em que lista ordenadamente as 500 empresas estado-unidenses com mais lucros.

cumprimento dos paradigmas do *Scrum* abafe todos os outros processo de controlo da qualidade de *software* ou da maturidade dos processos. Até os procedimentos provenientes das certificações *Capability Maturity Model Integration* (CMMI) [6] são possíveis em organizações com projectos que seguem o *Scrum*. Viljian Mahnic e Natasa Zabkar propõem um repositório de medidas e métricas (baseadas nos artefactos *Scrum*) que possibilita a monitorização constante e uma melhoria contínua da performance do processo de desenvolvimento de *software* em projectos que seguem o *Scrum* [Mahnic & Zabkar, 2008]. Mesmo a certificação máxima do CMMI (nível 5) é possível, segundo um dos co-autores do *Scrum* – Jeff Sutherland. Este recorda que o CMMI suporta institucionalização através de práticas genéricas associadas a processos de qualquer área. Assim, um conjunto de práticas genéricas estão redigidas no seu artigo de 2008, adaptadas a projectos geridos por métodos ágeis e que conseguem aumentar a performance dos mesmos, enquanto mantêm a conformidade com o nível 5 de CMMI [Sutherland & Johnson, 2008].

O *Scrum* também consegue suportar *frameworks* que permitam testar a correcta implementação dos requisitos e tarefas executadas pela equipa. Através de modelos *Unified Modeling Language* (UML)[7], um conjunto de investigadores alemães propõe uma metodologia, que conjuga os requisitos dos projectos de *Scrum* (denominados *User Stories*) com as técnicas de modelação ágil e de testes baseados em modelos (*model-based testing*) [Löffler et al., 2010].

Limitações. Como referido anteriormente, existe um conjunto de características no *Scrum*, disruptivas com os métodos tradicionais, e que lhe conferem um conjunto de vantagens importantes para enfrentar o grande flagelo na área de projectos em tecnologias de informação – o fracasso nos projectos de desenvolvimento de *software* (Tabela 1.1). Apesar dos testemunhos positivos e bons resultados alcançados, existem detalhes que o *Scrum* ainda não é capaz de lidar e que podem tornar-se num obstáculo intransponível para a sua utilização em determinados contextos.

[6] CMMI (*Capability Maturity Model Integration*) é uma abordagem para melhorar os processos, que providencia às organizações guias de orientação e elementos fundamentais para tornar os seus processos mais eficientes, o que garante a melhoria da sua performance. Informações disponíveis em: http://www.sei.cmu.edu/cmmi/

[7] UML (*Unified Modelling Language*) é um linguagem *standard* para modelação na área da Engenharia de *Software*. Mais informações podem ser encontradas em: http://www.uml.org

Em primeiro lugar, o *Scrum*, nos seus principais referenciais, não menciona uma única vez nenhum procedimento, protocolo ou processo relacionado com o desenvolvimento do produto. Não são tecidos comentários ou conselhos que preparem as equipas para desenvolver, na prática, código com elevada qualidade ou sem erros. Na realidade, a abordagem metodológica do *Scrum* apenas indica como a equipa deve ser gerida a um alto nível, focalizando-se na relação entre os intervenientes no projecto e no compromisso entre eles. Este facto pode levar algumas organizações e equipas que adoptem o *Scrum* a não aplicarem outros referenciais mais virados para a vertente da qualidade do *software*. Neste sentido, existem diversos exemplos de aplicação do *Scrum* com outras práticas, estas sim com maior foco no produto desenvolvido, como por exemplo o XP ou algumas áreas de processo do CMMI.

Outro problema no *Scrum* é o tempo de adaptação de uma nova equipa (ou uma equipa que, já tendo tido a experiência de trabalhar em conjunto, nunca adoptou o *Scrum*) até conseguir render o esperado. O *Scrum* confere à equipa grande poder de definição dos seus próprios objectivos e necessita de uma comunicação, conhecimento mútuo e comprometimento entre todos os elementos da equipa. Assim, é natural que sejam necessários alguns ciclos até a equipa atingir uma produtividade e maturidade suficientes para desenvolver eficientemente um produto.

Por outro lado, a dependência do *Scrum* na presença assídua e construtiva do cliente (no papel de *Product* Owner) pode tornar impossível a utilização desta abordagem, nos casos onde o cliente não está disponível ou não tem capacidade de gerir, testar e prioritizar o trabalho realizado. Nesta situação, a organização deve encontrar um responsável alternativo pelo produto, externo à equipa de desenvolvimento, com capacidade e responsabilidade para conduzir a equipa ao sucesso esperado.

Finalmente, o *Scrum* pode induzir alguma dificuldade no controlo, a longo prazo, do orçamento e do planeamento temporal do projecto. O facto dos planeamentos serem feitos *Sprint* a *Sprint* e o *Backlog* poder ser dinâmico e alterável, pode dificultar uma fotografia geral do projecto, principalmente naqueles que são mais longos. Nestes casos, a criação de um *Backlog Burndown Chart* pode ser útil, verificando e medindo a velocidade média da implementação da equipa ao longo das *Sprints,* ajudando assim na previsão dos custos e da data da finalização do projecto.

2.2.1 Os eventos do *Scrum*

Iteratividade. Uma das principais inovações nos métodos ágeis de desenvolvimento de *software* é a fragmentação do trabalho de um projecto. O *Scrum* não foge à regra, compondo ciclos no projecto que devem originar uma versão do produto, potencialmente comercializável. A esses ciclos de desenvolvimento são chamados *Sprints*.

Sprints. Têm duração variável e adaptável de projecto para projecto. Relatos indicam uma duração no intervalo de uma semana a um mês, em certos casos até mais. No final desta caixa temporal, um incremento de produto potencialmente comercializável é criado [Schwaber & Sutherland, 2011]. No entanto, as *Sprints* devem ser relativamente curtas para permitir a maior proximidade com o cliente, para despoletar a avaliação do trabalho efectuado mais frequentemente, de modo a perceber o rumo do projecto ou a corrigir eventuais desvios no produto. Caso exista algum problema com o produto, os riscos e fração temporal até ser lançada uma nova versão limitam-se ao máximo do tempo da *Sprint*.

Durante a *Sprint* existem vários fundamentos que devem ser respeitados, nomeadamente:

- Não devem ser feitas alterações que alterem a meta da *Sprint*;
- A composição da equipa não deve ser alterada;
- Os objectivos e o nível esperado de qualidade na realização das tarefas não deve diminuir.

Isto significa que a equipa, no início da *Sprint*, sabe o que a espera e deve sentir que tem todo o apoio e condições para a terminar com sucesso.

O próprio nome tem um simbolismo específico. Significa que os integrantes da equipa vão colocar todos os seus esforços neste curto espaço temporal de maneira a conseguir os objectivos propostos, comprometendo-se a terminar tudo o que foi planeado (fora algum desvio de maior, que pode levar ao cancelamento da *Sprint*).

As *Sprints* têm um tempo pré-definido, que em caso algum deve ser extendido. Se na aproximação ao final da *Sprint* os objectivos forem manifestamente impossíveis de cumprir, existem actores (nomeadamente o *Scrum Master* ou o *Product Owner*) que podem ordenar o seu cancelamento. Esta consequência tem sempre impactos negativos

na equipa, dado que foram estes quem se comprometeram perante os *stakeholders* que conseguiriam terminar as tarefas propostas e estimadas pela equipa para esta *Sprint*. Adicionalmente, se existir alguma alteração abrupta de contexto que torne os objectivos da *Sprint* obsoletos, o cancelamento será igualmente uma opção válida. Em cada *Sprint* existe um número de eventos que ocorrem obrigatoriamente: o planeamento (*Sprint Planning*), a revisão diária (*Daily Scrum*), a revisão (*Sprint Review*) e ainda a retrospectiva (*Sprint Retrospective*).

Sprint Planning. Todos os ciclos começam com o planeamento. É neste momento que todos os elementos da equipa se juntam e revisitam o *Backlog,* de maneira a ver quais as *User Stories* mais urgentes a necessitarem de atenção. Este evento é dos mais importantes no *Scrum*, é aqui que se faz o planeamento para o próximo ciclo de trabalho e, ainda mais importante, é aqui que se sela o compromisso entre a equipa e o *Product Owner* sobre o trabalho que irá ser realizado nessa *Sprint*.

Para que as equipas não se desorientem na discussão sobre o trabalho a efectuar, é aconselhável que a reunião tenha uma duração máxima de 4 horas para uma *Sprint* de duas semanas, sendo que a duração pode aumentar/diminuir proporcionalmente com a duração da *Sprint*.

Tipicamente, esta reunião é dividida em duas partes distintas. Na primeira metade selecionam-se as *User Stories* a completar (dento do *Product Backlog* previamente ordenado por prioridade pelo *Product Owner*) e na segunda parte define-se a maneira como se vai implementar cada *User Story*, definindo tarefas específicas e estimando o tempo necessário para cada uma delas. Nas estimativas para cada tarefa, é possível utilizar-se uma relação matemática entre o tempo estimado da realização da tarefa no melhor caso possível (tudo funciona à primeira, sem defeitos a resolver), no caso mais provável (se poderá ser necessário fazer alguma investigação prévia, se originar alguns defeitos, etc.) e no pior caso possível (necessária muita pesquisa para a implementação, originar muitos defeitos ou incompatibilidade imprevista com outras partes do produto). Estas estimativas devem ter em consideração a complexidade da tarefa, a utilização de novas tecnologias ou de tecnologias experimentadas anteriormente pela equipa, a probabilidade de gerar defeitos graves ou problemas de incompatibilidade entre componentes ou versões do produto, entre outros. Apesar de todas as estimativas, é prática comum que as duração das tarefas não ultrapassem os dois dias (ou 16 horas),

de maneira a ser mais fácil de acompanhar e perceber o estado da implementação diariamente [Cervone, 2011]. Para além da estimativa, é muito importante ficar definido entre todos os elementos da equipa e o *Product Owner* as características fundamentais para se considerar uma tarefa completa, ou utilizando a gíria do *Scrum*, qual a definição de *Done* para cada *User Story*. Ao conjunto de tarefas definidas para aquela *Sprint* é chamado *Sprint Backlog.*

***Daily Scrum*.** É um evento que ocorre diariamente e onde cada elemento da equipa responde a três questões:

- O que foi realizado desde a última *Daily Scrum*?
- O que será feito até à próxima *Daily Scrum*?
- Existe algum obstáculo actualmente para a realização da tarefa a decorrer?

Esta reunião é utilizada pelo *Scrum Master* para perceber a direcção e a velocidade da equipa na *Sprint* que está a decorrer, e para os restantes elementos da equipa se aperceberem do estado das tarefas e quais as dificuldades percepcionadas pelo elemento que a está a resolver. Através deste método, é possível que qualquer elemento da equipa consiga ter uma noção das tarefas em realização e qual o contexto das mesmas, inclusivamente se é necessário a sua intervenção ou ajuda. Neste evento, a *Sprint Backlog* é revista e serve de auxiliar à condução da reunião.

Para a *Daily Scrum* ser eficaz, a sua duração não deve exceder os 15 minutos, deve ser sempre realizada no mesmo local e hora e os elementos devem permanecer em ortostatismo. Através destas condições, cria-se uma hábito eficaz porque a equipa se apercebe diariamente da sua distância para atingir o objectivo da *Sprint,* e eficiente, porque este evento torna-se rotineiro e rápido, não tendo um impacto temporal significativo num dia de trabalho.

Nesta reunião devem estar apenas a equipa e o *Scrum Master*. O *Product Owner* não deve estar presente pois esta é uma reunião operacional, onde se discute detalhes práticos da implementação e execução das tarefas. É aqui que os elementos da equipa expõem as suas preocupações e dúvidas para o resto dos companheiros. Caso o *Product Owner* estivesse presente, algumas duvidas ou obstáculos poderiam não ser colocados, com receio de alguma avaliação ou desconfiança nas suas capacidades por parte do *Product Owner*.

Esta reunião é de extrema importância e a sua não execução de modo diário pode colocar em causa o sucesso do projecto. Promove-se a rápida tomada de decisões, a partilha de conhecimentos entre a equipa e a remoção diária de obstáculos ao desenvolvimento do produto, tudo características fundamentais de um projecto gerido através de uma metodologia ágil.

Sprint Review. Consiste na reunião que acontece no final da *Sprint*, de avaliação e revisão ao trabalho efectuado durante esse ciclo de desenvolvimento. Está presente a equipa de desenvolvimento, o *Scrum Master*, o *Product Owner* e qualquer outro *stakeholder* que assim o deseje (Clientes, Gestão de Topo, entre outros). A equipa demonstra o trabalho executado e as *User Stories* completas, sendo feita uma análise pelo *Product Owner* que, à luz do estabelecido na reunião de planeamento, dá o seu aval e considera (ou não) que a *User Story* respeita a definição de *Done* acordada no *Sprint Planning*.

Neste encontro, podem ser discutidos quaisquer outros assuntos relacionados com a *Sprint*, a equipa pode discutir o trabalho futuro, as soluções técnicas encontradas para a implementação dos requisitos e os *stakeholders* podem questionar a equipa sobre aspectos práticos do produto. Consoante a avaliação e o trabalho efectuado, o *Product Owner* pode reordenar ou ajustar o *Product Backlog* para se adaptar a novos contextos.

***Sprint Retrospective*.** Traduzido do inglês, pode ser apelidada de retrospectiva da *Sprint.* Nesta ocasião, a equipa junta-se ao seu *Scrum Master* e discute a maneira como a *Sprint* correu em termos operacionais. Aqui se pode discutir a maneira como as reuniões ocorridas decorreram, planear formas de tornar a aplicação do *Scrum* mais eficiente ou analisar outras causas externas à equipa mas que podem estar a condicionar o desempenho da mesma. Desta reunião podem surgir um conjunto de melhorias que podem ser integradas na próxima *Sprint* de maneira a tornar a equipa mais eficiente ou melhorar a qualidade do produto desenvolvido. É mais uma etapa fundamental na crónica demanda das metodologias ágeis - inspeccionar e adaptar.

Grooming. Também apelidada de *Backlog Refinement Meeting*, é uma reunião que não é obrigatória ocorrer em todas as *Sprints*. A equipa junta-se ao *Scrum Master* e ao *Product Owner* e ajuda a escrever os requisitos do produto em forma de *User Stories*.

Esta reunião é importante pois serve como plataforma de entendimento entre o *Product Owner* e a equipa sobre o trabalho que ainda falta realizar sobre o produto. Os itens que são vagos ou abstractos são clarificados e divididos em *User Stories* que representem pequenos incrementos de valor ao produto. Além da clarificação dos requisitos, a equipa ajuda também o *Product Owner* a perceber o grau de complexidade de implementação de cada *User Story*, atribuindo-lhe *Story Points*. Os *Story Points* podem ser valores numéricos sequenciais, ordens de grandeza crescentes (por exemplo, os números da sequência de *Fibonacci*) ou outro tipo de representação simples. De modo a estimar mais facilmente todas as *User Stories*, as equipas devem manter uma *User Story* padrão e de exemplo com uma pontuação concordante entre todos, para servir de comparação. Nas estimativas das novas *User Stories*, os elementos da equipa devem discutir entre si e chegar a um consenso sobre o valor de complexidade a atribuir a cada uma.

Estas estimativas de alto nível servem para o *Product Owner* perceber, com o decorrer dos ciclos, quantos *Story Points* consegue uma determinada equipa fazer durante uma *Sprint* (métrica chamada *Sprint Velocity*). Através dessa velocidade média de implementação de *User Stories*, juntamente com os itens do *Product Backlog* estimado, o *Product Owner* consegue escolher melhor as *User Stories* que devem avançar no início de uma nova *Sprint* de maneira a conseguir um incremento real e completo ao produto.

2.2.2 Os actores do *Scrum*

Scrum Master. É o responsável pela implementação e utilização da abordagem metodológica do *Scrum* num projecto, assim como a sua configuração de modo a tornar a equipa o mais eficiente possível. É diferente do tradicional Gestor de Projecto porque não tem poder decisório sobre o trabalho a executar ou o produto que se está a implementar. Tem como função conduzir todos os eventos do *Scrum* (*Daily, Review* e *Retrospective*) e a manutenção do *Product Backlog* e do *Sprint Backlog* em estreita comunicação com o *Product Owner* e a equipa, respectivamente. Através da monitorização do trabalho da equipa e da manutenção dos artefactos, o *Scrum Master* deve ser capaz de fazer um diagnóstico conciso sobre a *performance* da equipa numa determinada *Sprint*, avisando os outros actores caso exista alguma ocorrência que possa colocar em causa o sucesso da *Sprint*.

Apesar de não ter autoridade sobre o trabalho e o produto, este elemento deve saber incorporar o papel de líder, respeitado pela equipa e pela gestão de topo (assim como pelo *Product Owner*) dado que ele é a ponte entre os operacionais e os gestores. Cabe também ao *Scrum Master* a resolução de todos os impedimentos de carácter não técnico que a equipa e os seus elementos tenham, de maneira a conseguir focalizar os mesmos apenas na execução das suas tarefas.

Product Owner. Assume-se como responsável do produto, cabendo as decisões mais importantes relativo ao rumo do projecto, de maneira a conseguir maximizar o retorno do investimento (ROI). Tem poder decisório sobre os requisitos a implementar e qual a sua prioridade (através da gestão e da re-prioritização do *Product Backlog*), assim como sobre a configuração da equipa. Neste aspecto, o *Product* Owner pode ser visto com algumas responsabilidades que nos métodos tradicionais de gestão de projectos de *software* se podem mapear com as responsabilidades do *Project Manager*, nomeadamente em relação aos requisitos e à relação com os *stakeholders* [Collaris et al., 2010]. Deve comunicar preferencialmente com o *Scrum Master* sobre a visão que tem para o produto, para que este último adapte o processo *Scrum* aos seus objectivos. Pode ser assessorado por outros *stakeholders* (como clientes ou gestão de topo) de maneira a avaliarem e aceitarem cada incremento realizado e demonstrado na *Sprint Review*. Para se desempenhar este papel não é obrigatório um conhecimento técnico aprofundado do produto e das tecnologias implementadas, sendo este papel muitas vezes desempenhado por profissionais da área da gestão ou do *marketing* [Schatz & Abdelshafi, 2005]. Na relação com o produto e com a equipa de implementação, o *Product* Owner é o único com autoridade para agir e tomar decisões. Também existem alguns casos de *Product Owner* que são simultaneamente os clientes do produto.

Equipa de Desenvolvimento. Tal como o nome indica, são os executantes técnicos do projecto, aqueles que implementam as funcionalidades do produto e, no fundo, a força motriz essencial para o sucesso. Num projecto gerido pela abordagem metodológica do *Scrum*, o número de pessoas da equipa de desenvolvimento deve oscilar entre os 3 e os 10. Mais do que a dezena de intervenientes pode pôr em causa a eficiência de uma abordagem metodológica que assenta na comunicação e troca de

conhecimentos entre todos os elementos da equipa, além de requerer uma coordenação dos eventos e dos artefactos que poderia ter um custo demasiado alto.

Ao contrário de outras metodologias (como RUP por exemplo), no *Scrum* apenas existe um papel único na equipa: o de *developer.* Todos os intervenientes devem ter capacidade para entender o trabalho efectuado em qualquer âmbito da implementação do produto, embora seja natural que existam elementos com alguma especialização e que estejam mais rotinados para determinadas tarefas. No entanto, a imputabilidade de todo e qualquer tipo de tarefa é da equipa e não de algum elemento individual [Cohn, 2003].

2.2.3 Os artefactos do *Scrum*

Product Backlog. É o local onde estão ordenados e documentados todos os requisitos funcionais do produto, tanto os implementados como os por implementar. Artefacto de responsabilidade do *Product Owner* e gerido com a ajuda do *Scrum Master*, deve estar sempre visível e disponível para consulta da equipa ou de outros *stakeholders* interessados.

Os requisitos estão apresentados, tipicamente, em formato de *User Stories*. Uma *User Story* é uma frase em que se especifica o que se deseja ser implementado e qual o actor que se relaciona com esse requisito. Para cada *User Story* deve ser definido a sua medida de complexidade de implementação (normalmente definida como *Story Points*) e a sua definição de *Done* (características que permitem ao *Product Owner* considerar uma determinada *User Story* como implementada ou não).

Caso o esforço para a implementação de uma determinada *User Story* for maior do que o esforço de 2/3 elementos em 2/3 dias, a *User Story* deve provavelmente ser dividida em várias, de maneira a facilitar o controlo e evitar perder o foco diário ao que se tem de fazer. Durante a reunião de planeamento, para cada *User Story,* irão ser definidas as tarefas que permitem a sua implementação completa e que podem ser executadas por qualquer membro da equipa. Essas tarefas devem ter duração máxima de 16 horas e deve ser re-estimado o seu esforço em falta diariamente.

Sprint Backlog. É o conjunto de itens do *Product* Backlog - *User Stories* (e respectivas tarefas) - que a equipa se comprometeu a implementar para uma determinada *Sprint*. A sua criação é da responsabilidade exclusiva da equipa durante a

segunda fase da reunião de planeamento, embora a sua manutenção possa ser feita pelo *Scrum Master*. Normalmente de formato simples e de legibilidade imediata, tem três secções: tarefas por fazer, tarefas em execução e tarefas concluídas. Como as tarefas têm uma estimativa temporal associada, a qualquer altura o *Scrum Master* pode ter uma panorâmica completa sobre o trabalho que já foi executado e aquele que falta executar. Este artefacto é revisitado durante a *Daily Scrum* para que toda a equipa tenha a percepção do que ainda é esperado de si durante a *Sprint*.

Sprint Burndown Chart. Artefacto que está intrinsecamente associado ao *Sprint Backlog*, é a visualização em gráfico da quantidade (tipicamente em horas) de trabalho que falta executar até ao final da *Sprint*. Relaciona as horas de trabalho disponíveis (multiplicando o número de horas úteis até ao final da *Sprint* pelo número de elementos da equipa) e o somatório das horas das tarefas que ainda falta realizar. Tal como o *Sprint Backlog*, este gráfico é visualizado pela equipa em cada *Daily Scrum*. Embora represente o esforço despendido pela equipa em relação ao trabalho estimado, podendo expor algum atraso, a sua interpretação deve ser efectuada apenas pela equipa. Existem outras estatísticas que podem ser analisadas paralelamente, como o ritmo de trabalho diário, o acumulado de horas em atraso, entre outros.

Caso existam outros *stakeholders* a visualizarem o gráfico e a tomarem medidas correctivas que impactem com o normal funcionamento da equipa, o *Scrum Master* pode optar por colocar de parte a utilização do *Sprint Burndown Chart*.

É nesta relação maleável entre eventos, actores e artefactos que está uma das forças do *Scrum.* Esta abordagem dá a liberdade para adaptar o tamanho da equipa às necessidades decorrentes (máximo 10 pessoas), permite a criação de ciclos de desenvolvimentos com duração flexível a cada organização, possibilita a avaliação permanente do produto através das *Sprint Review* e promove a ajuda mútua e partilha do conhecimento entre os elementos da equipa. Detalhes de implementação do *Scrum* são únicos para cada projecto, mas a panorâmica geral é sempre idêntica e pode ser visualizada na Figura 2.1.

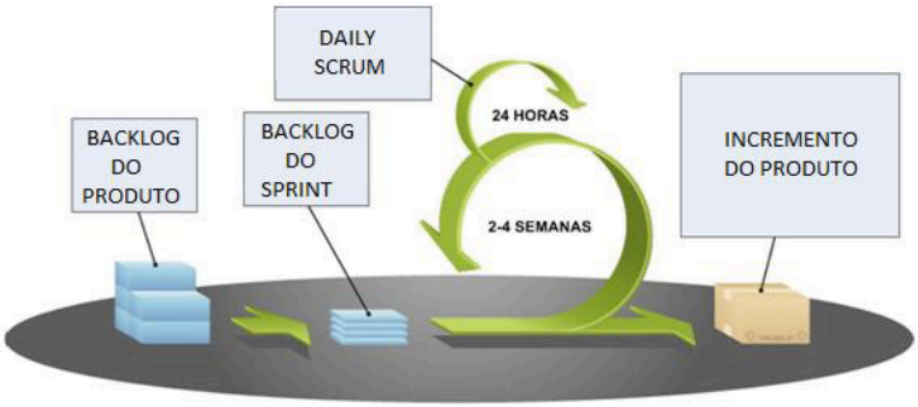

Figura 2.1 – Visão geral do fluxo do *Scrum*. Fonte: [Rose & Mello, 2010]

2.3 User Stories

Definição. Uma *User Story* é uma caracterização alto nível de um requisito, contendo apenas a informação necessária para que os programadores do projecto consigam perceber claramente o que é pretendido implementar, capacitando-os facilmente de estimar a sua complexidade e o esforço requerido para a sua implementação [Ambler & Lines, 2012].

Tipicamente, uma *User Story* deve ser pequena (uma frase concisa) e deve conter três características da funcionalidade a querer implementar: "quem", "o quê" e "porquê".

A primeira particularidade que é necessário interiorizar quando se interpreta uma *User Story* é o elemento (actor) a quem a funcionalidade se destina. Na disciplina da Engenharia de *Software*, o actor representa um papel praticado por uma entidade que está directamente relacionado com o sistema a implementar. Os actores podem ser representados por humanos, por *hardware,* ou até mesmo por outros *softwares* que, embora estejam relacionados com o sistema a implementar, não são parte integrante. Sabendo o actor a que se refere, um programador deve conseguir interiorizar os seus objectivos e as suas responsabilidades para com o sistema, de modo a que a implementação seja mais direcionada para aquilo que realmente interessa para o actor.

Depois de se saber com quem está relacionada uma determinada actividade, é necessário compreender em que é que consiste essa mesma actividade. O "quê" da *User Story* é um dado fundamental para delimitar o que é pretendido para o determinado actor. Esta característica é o que possibilita a estimativa em termos de complexidade ou de esforço de implementação de uma *User Story*.

Por fim temos o "porquê". Embora seja um campo que por vezes é menorizado na sua importância, pode ser extremamente útil e relevante. Com este detalhe, torna-se mais simples às equipas focalizarem-se na verdadeira razão pela qual uma determinada actividade é importante para um determinado actor, contextualizando a sua importância.

De seguida estão alguns exemplos genéricos de possíveis *User Stories*:

- "Como Administrador de Sistemas, quero saber o número de utilizadores registados, de modo a conseguir gerir eficazmente a plataforma";
- "Como utilizador, quero ter acesso ao meu saldo, de maneira a saber quanto dinheiro tenho na conta".

I.N.V.E.S.T. Apesar de todas as *User Stories* possuírem um sujeito, uma actividade e um motivo para uma determinada actividade ser necessária, não existe uma garantia que todas as *User Stories* escritas desta maneira sejam úteis para as equipas de desenvolvimento de *software*. Existem *User Stories* que, apesar de semanticamente correctas, podem não representar uma funcionalidade que pode ser escalada numa *User Story.* Deste modo, as *User Stories* devem respeitar as seguintes premissas:

- *(I)ndependent* – Independentes. As *User Stories* devem ser independentes umas das outras. Isto significa que elas devem estar escritas de uma maneira que qualquer uma pode ser implementada a qualquer altura, não estando dependentes da implementação anterior ou posterior de qualquer outra *User Story*;
- *(N)egotiable* – Negociável. Baseando-se no pressuposto de que o *Scrum* potencia o trabalho de equipa e a colaboração/comunicação entre todas as partes envolvidas, também as *User Stories* podem ser sujeitas a alterações desde a altura em que são criadas até à sua implementação. Diversos tipos de profissionais podem ter diferentes entendimentos sobre um determinado requisito e um entendimento deve ser alcançado entre a equipa de implementação e o cliente, para conseguir um requisito valioso para o cliente e executável pela equipa;
- *(V)aluable* – Valioso. Todas as *User Stories* implementadas devem significar um acréscimo tangível para o produto. Para se conseguir esse objectivo, é necessário a colaboração entre o cliente e a equipa de desenvolvimento para perceber se o que se vai implementar vai acrescentar valor ao produto

ou, pelo contrário, pode ser um factor de geração de problemas para o produto ou para o utilizador final. Apesar do foco no cliente ser importante, por vezes é necessário pensar em factores não funcionais que, não sendo perceptíveis directamente ao utilizador, beneficiam-no ao induzir melhorias no funcionamento do produto. *User Stories* que mapeiam requisitos não funcionais são exemplos de áreas de trabalho, que não gerando funcionalidades novas para o produto, podem ser extremamente importantes na consolidação do *software* no mercado. Além do mais, com o recurso à prioritização do *Product Backlog,* as *User Stories* que representem funcionalidades de menor valor para o cliente podem ser "empurradas" para os últimos lugares, assegurando assim que as funcionalidades fundamentais são implementadas primeiro;

- *(E)stimable*. Estimáveis. Um dos pontos importantes no *Scrum* é que os *stakeholders* interessados num determinado projecto podem ter a capacidade de saber, em tempo real, qual o esforço necessário para a implementação de determinada *User Story*. Essa percepção pode ajudar na prioritização do *Backlog* e na decisão sobre o que se vai implementar de seguida. Por vezes, é preferível implementar algumas funcionalidades que, não sendo tão complexas, podem ser implementadas num menor espaço de tempo, de modo a serem colocadas em produção o quanto antes. O evento de *Grooming* é importante na aferição da estimabilidade das *User Stories*, dado que é nesse momento que a equipa atribui um grau de complexidade (geralmente directamente proporcional às horas de esforço necessárias para a sua implementação), que é importante para o *Product Owner* (ou para o cliente) na hora da decisão sobre qual a próxima implementação;
- *(S)mall*. Curtas. Outro factor fundamental para eficiência do processo é projectar *User Stories* que, mantendo o valor para o produto, sejam possíveis implementar num espaço de tempo relativamente curto. Esta medida temporal é diferente de projecto para projecto, mas é importante que o esforço de implementação de uma determinada *User Story* seja inferior à duração de uma *Sprint.* Deste modo, a equipa consegue não perder o foco relativamente ao que é realmente importante, problema recorrente nas metodologias onde esta granularidade não é aplicada na

definição dos requisitos a implementar. Caso existam *User Stories* de uma complexidade tal que a sua execução dure mais do que o tempo da *sprint*, isso pode indiciar que pode ser decompostas em *User Stories* mais pequenas mas que continuem alinhadas com os princípios INVEST, fazendo com que essa determinada *User Story* possa ser transformada num *Theme* ou num *Epic* (*User Stories* de alto nível que são compostas por *User Stories* mais granulares);

- *(T)estable*. Testável. Um detalhe fundamental para cada *User Story* é que ela deve ser escrita e implementada de maneira a poder ser testada e que, assim, seja assegurada ao cliente que a implementação foi bem conseguida. É um detalhe que é esquecido frequentemente pelas equipas, que não reservam tempo da sua *Sprint* para preparar convenientemente a problemática do teste de *software*. Os testes devem ser especificados ainda antes da implementação da *User Story* pela equipa e, se possível, devem ser automatizados e corridos sempre que se introduza alguma alteração no código do produto. A execução dos testes sobre a implementação de uma determinada *User Story* pode ocorrer durante o *Sprint Review,* ajudando o *Product Owner* a decidir se a condição de *Done* foi atingida, avalizando assim o fecho da implementação da *User Story* com sucesso.

A geração de *User Stories* que sigam os princípios INVEST são, por si só, uma segurança que a sua implementação tenha todas as condições para ser executada correctamente. É importante que todos os *stakeholders* tenham bem presente estes princípios na altura da escrita das mesmas, de modo a facilitarem o trabalho às equipas de implementação.

Representação. Existem diversas formas de representar e armazenar as *User Stories* e respectivas tarefas, para que estejam permanentemente disponíveis para consulta por todos os intervenientes no processo de desenvolvimento do *software.* Como a maioria das plataformas que dão suporte a todo o ciclo de vida dos produtos estão informatizadas, é natural que o meio favorito para a geração e gestão de *User*

Stories durante uma *Sprint* seja com a utilização de um qualquer componente de *software*.

No entanto, e dado que muitas vezes as equipas de desenvolvimento ágeis estão todas localizadas no mesmo espaço – de modo a intensificar e encorajar a comunicação e a troca de conhecimentos – existem diversos exemplos de *Scrum Master* que optaram por dispor o seu *Sprint Backlog (User Stories* e respectivas tarefas) num local físico e visível. Apesar dos constantes esforços de integração de todos os detalhes relacionados com o desenvolvimento de *software* numa única ferramenta, de maneira a conseguir gerar métricas por projecto para suportar modelos de maturidade, ainda existem equipas que acham útil continuar a ter um local reservado para a percepção do estado da *sprint* a todo o momento [Boehm, 2007]. O *feedback* visual proporcionado por uma parede preenchida por cartões ou *post-its* ajuda as equipas a manterem o foco e a criarem uma cultura de comprometimento para com os objectivos da *Sprint.* Não obstante, é sempre necessário representar o estado da *Sprint* numa ferramenta onde a restante gestão e *stakeholders* possam ter acesso [Berczuk, 2007].

Figura 2.2 - Exemplo real de um quadro *Scrum* físico [8]

Tipicamente, as equipas preveem três estados para cada tarefa de uma determinada sprint: Por fazer (*ToDo)*; A fazer (*On-Going*); Feito (*Done).* Estas tarefas transitam de estado em estado consoante a sua implementação e respectivos testes já terem terminado ou não. Apesar de, para efeitos visuais, as tarefas (e respectivas *User Stories*) serem consideradas feitas (ou *Done)* no final da sua implementação, apenas o

[8] Imagem real, retirada do Blog CIGAM – *Scrum* INFRA-Blog - http://cigaminfra.wordpress.com/2009/11/

Product Owner na *Sprint Review* pode efectivamente considerar uma determinada funcionalidade implementada e validada.

***User Stories* de Suporte**. Embora o foco dos clientes e das equipas esteja em acrescentar novas funcionalidades ao produto a cada *Sprint* que passa, existem tarefas (que podem ser mapeadas eventualmente em *User Stories*) ao qual é necessário reservar algum tempo e preocupação na implementação de um produto. Cuidados a ter com os requisitos não funcionais do produto, como escalabilidade, segurança, portabilidade, entre outros, são da maior relevância. Por vezes, o foco excessivo em implementar regularmente diversas funcionalidades novas pode fazer com que o produto, apesar de estar capacitado a executar diversas acções, não esteja capacitado para propor aos utilizadores uma experiência condizente com o número de funcionalidades implementadas. É assim necessário negociar com o cliente um conjunto de requisitos não funcionais para balizarem a implementação de novas funcionalidades.

Além desse tipo de requisitos, também os defeitos encontrados na fase de *deployment*, testes ou manutenção precisam de ser endereçados e convertidos em *User Stories* (ou pelo menos em tarefas), para serem colocados no *Backlog*. Tal como qualquer outra *User Story*, depois do defeito estar documentado e detalhado, cabe à equipa especificar um conjunto de tarefas (e respectivo esforço de implementação) para o *Product Owner* decidir a altura mais oportuna de lançar na *Sprint* tarefas ou *User Stories* relacionadas com a correcção de defeitos de *software.* Por vezes, a criticidade dos defeitos encontrados pode ser tal que, para uma nova *Sprint*, sejam estas as *User Stories* mais prioritárias e as primeiras a serem lançadas para execução. Inclusivamente, a duração das *Sprints* pode ser encurtada de forma a conseguir obter uma revisão dos defeitos corrigidos mais rapidamente, gerando assim luz verde para colocar uma nova versão do *software* em fase de produção mais rapidamente. ´

2.4 O *Scrum* Distribuído

Paralelamente à utilização do *Scrum* em pequenas empresas ou em pequenos projectos, a indústria começou a ter a percepção dos bons resultados que as equipas que aplicavam *Scrum* iam obtendo. Nesse sentido, começaram a estudar as técnicas do *Scrum* e a tentar adaptar os eventos, actores e artefactos para equipas que estivessem

distribuídas geograficamente, ou para múltiplas equipas que trabalhassem sobre o mesmo produto.

Assim, Jeff Sutherland, um dos mentores da abordagem metodológica *Scrum*, classifica o *Scrum* Distribuído em dois tipos distintos [Sutherland et al., 2006]:

- *Scrum* of *Scrums* – Onde existem diversas equipas que trabalham para o mesmo produto e em que os elementos de cada equipa estão agrupados geograficamente. Neste caso, existem elementos (*Product Owner*, *Scrum Master* ou um elemento da equipa de implementação) que se encontram regularmente de maneira a conseguir integrar o trabalho das respectivas equipas [Cristal et al., 2008].
- *Scrum* Totalmente Integrado (do inglês: *Totally Integrated Scrum*), onde os elementos das equipas estão distribuídos por geografias diferentes, o que proporciona uma adaptação de toda a abordagem metodológica *Scrum* de maneira a conseguir tornar a equipa funcional e eficiente [Paasivaara et al., 2008].

Paralelamente, o *Scrum* também foi experimentado em projectos que envolviam várias organizações diferentes a tentar cumprir e implementar o mesmo produto, com bons resultados em projectos de dimensões limitadas [Dingsøyr et al., 2006].

Tal como a própria abordagem metodológica do *Scrum*, estas variantes foram testadas e criadas em ambiente industrial, com experiências *ad-hoc* e melhorias constantes. Na literatura da área, apenas existem testemunhos de organizações que tentaram adaptar o *Scrum* para as suas equipas, não existindo um referencial completo e transversal sobre como implementar o *Scrum* em projectos com equipas distribuídas. Aspectos como sincronização de sprints, plataformas de entendimento entre várias equipas que trabalham sobre o mesmo produto, cadência de reuniões, entre outros, continuam a ser tópicos onde existe uma vasta área de progresso para percorrer.

2.5 Conclusão

O paradigma da gestão de projectos de *software* está a mudar rapidamente, com a introdução de metodologias ágeis numa extensão que percorre desde os projectos pequenos [Rising & Janoff, 2000] até aos projectos e organizações de grande dimensão

[Begel & Nagappan, 2007; Scotland & Boutin, 2008]. Este novo paradigma permite a adaptação das empresas do ramo das Tecnologias da Informação, capacitando-as a desenvolver com maior celeridade, com menos formalismos, acompanhando a velocidade de evolução das novas tecnologias e de novas necessidades por parte do cliente [Baskerville et al., 2003].

O *Scrum*, como uma das faces mais visíveis dos métodos ágeis de desenvolvimento de *software,* capacita as equipas a planearem, implementarem e testarem o produto e o processo de gestão da própria equipa, adaptando o processo aos seus intervenientes, de modo a conseguir extrair o máximo de cada um deles. Simultaneamente, o *Scrum* possibilita um canal de comunicação contínuo e aberto entre o cliente final e o produto a ser desenvolvido, facilitando assim mudanças e alterações em tempo real, aproximando cada vez mais o produto gerado das necessidades reais do cliente.

Apesar de não haver referência expressa à necessidade de documentação formal para se iniciar a fase de desenvolvimento do produto, existem vantagens na delineação atempada de arquitecturas físicas e de componentes, de maneira a balizar todo o trabalho que, embora iterativo, é melhor consubstanciando quando efectuado à luz de uma arquitectura pré-definida. Este facto é ainda mais importante quando os produtos são complexos e implementados por várias equipas em paralelo. Abre-se assim um espaço de discussão de novos procedimentos que juntem os requisitos do *Scrum* (*User Stories)*, e as arquitecturas lógicas necessárias para especificar um bom produto.

3. Das Arquitecturas Lógicas às User Stories

3.1 Introdução

Como foi descrito nos capítulos anteriores, as metodologias ágeis são utilizadas no contexto onde, por vezes, nem os próprios clientes conhecem os limites do sistema que pretendem. Frequentemente, inicia-se a implementação das soluções partindo apenas das características essenciais requeridas (vulgarmente conhecidas por *user needs* – necessidades de utilizador), esperando que a proximidade entre equipa de desenvolvimento e o cliente molde os requisitos secundários e não funcionais. Este *modus operandi* é claramente disruptivo em relação às metodologias tradicionais de desenvolvimento de *software*, suportados em modelos cascata, onde existe um grande esforço inicial na angariação de todos os requisitos e na modelação do referido sistema.

Passando para a plataforma da indústria de *software*, o paradigma está maioritariamente focado na qualidade do código escrito. Assim, as empresas (nomeadamente as mais pequenas) especializam-se e contêm um *know-how* técnico muito forte, descurando outras partes igualmente importantes na criação de um produto de *software*, como a gestão de requisitos ou a modelação de sistemas. Esta falha é também uma causa para a escolha das metodologias ágeis, que não necessitam de uma formalização complexa das etapas que tipicamente antecedem a implementação do *software.* Consequentemente, pode dar-se o caso das próprias empresas de *software* fazerem uma sub-contratação ou uma associação com outras entidades, estas últimas especializadas nas fases de gestão de requisitos e criação de modelos representativos dos produtos/serviços a construir, como é o caso dos centros de investigação das Universidades de competência tecnológica. Adicionalmente, no panorama actual português das pequenas organizações que criam *software*, uma das principais fontes de

rendimento provem de projectos co-financiados por fundos governamentais, onde as características e principais requisitos já são conhecidos à partida, e onde o contacto com o preponente do trabalho é difícil e nem sempre é regular, tal como exige uma metodologia ágil.

Actualmente, existem várias técnicas capazes de produzir arquitecturas lógicas partindo de um aglomerado de necessidades de utilizador ou requisitos, capazes de estruturar e agrupar esses mesmo requisitos em *packages* (aplicações). Essa representação facilita a percepção das interdependências entre os requisitos e permite observar pontos fulcrais na arquitectura de um sistema, pela observação dos componentes que comunicam esses pontos. Uma das metodologias mais utilizadas no panorama académico português é o *4-Step Rule Set* (4SRS), uma técnica que converte Casos de Uso (mais uma forma de representar requisitos funcionais) em Diagramas de Componentes, baseados em UML, representativos da arquitectura lógica do sistema. Embora os resultados da aplicação do 4SRS sejam um importante contributo para o trabalho das equipas de implementação de *software*, por si só não ajudam a estruturar o trabalho em unidades reconhecidas e facilmente integradas nas equipas que seguem abordagens metodológicas ágeis, como é o caso do *Scrum*.

Partindo do princípio de que a figuração de um sistema numa representação lógica (como um Diagrama de Componentes) é uma parte importante para a interpretação correcta das funcionalidades que são necessárias implementar, abre-se assim um espaço não explorado pelo meio académico, que permita a transição entre requisitos (estejam eles escritos apenas textualmente, com recursos a diagramas UML de Casos de Uso ou através de outras técnicas) e eventuais modelos para *User Stories*, adaptando assim as equipas que seguem metodologias ágeis aos projectos que possuem necessidades definidas *à priori*, e onde o contacto com as entidades adjudicantes pode ser difícil e bastante espaçado no tempo.

Apesar do distanciamento das perspectivas ágeis e da especificação de documentação formal, a focalização da equipa *Scrum* num conjunto de requisitos derivados de uma arquitectura lógica não é necessariamente um contrassenso com os princípios fundamentais das metodologias ágeis, desde que a presença da arquitectura contribua para a eficácia da implementação, não cause entropia ou atrasos substanciais devido à sua manutenção. Além disso, a não valorização das arquitecturas do sistema a desenvolver por parte das equipas pode trazer problemas de integração no *software,* dado que as *User Stories* geradas podem ter dependências sobre determinadas

componentes lógicas da arquitectura, razão pela qual este tipo de diagramas pode ter um importante papel no seio das equipas de desenvolvimento geridas pela aplicação do *Scrum* [Brown et al., 2010]. Esta consideração é tão mais valida quanto maior for a complexidade do produto a desenvolver ou o número de equipas a trabalhar em paralelo para um produto.

Para atingir essa conjunção potencialmente benéfica entre arquitecturas lógicas e *User Stories* é proposto um novo processo, chamado FLAUS (*From Logical Architectures to User Stories*), que combinado com o 4SRS (capaz de gerar diagramas lógicos representativos do sistema), é capaz de criar um conjunto de *User Stories* facilmente interpretáveis pelas equipas que seguem a abordagem metodológica do *Scrum*.

3.2 Arquitecturas Lógicas e o 4SRS

A modelação de uma arquitectura de um sistema a implementar, partindo de um conjunto de requisitos funcionais, é das actividades mais complexas e *ad-hoc* que existem na disciplina de Engenharia do *Software*. É um procedimento pouco formalizado, que recorre à intuição dos seus autores e que se reveste de uma criticidade extrema, pois o resultado do procedimento tem um impacto e uma influência enorme em todo o processo de implementação do produto em questão [Bosch & Molin, 1999; Machado et al., 2006].

As arquitecturas lógicas, como uma representação possível da arquitectura de um Sistema de Informação, podem ser descritas como uma especificação de um sistema, constituída por um conjunto de abstrações de questões específicas do problema/domínio que representam requisitos funcionais [Azevedo et al., 2009; Castro et al., 2002]. Estas representações são importantes para a validação dos requisitos elicitados em fases anteriores mas, por si só, não são garantia de que toda a informação necessária está identificada naqueles diagramas. Paralelamente, existe outra informação (textual ou em forma de diagrama) que é complementar àquela representada nos diagramas UML de arquitecturas lógicas e, sem ela, os *stakeholders* não possuem a informação total que lhes permitiria tomar decisões acertadas em relação à implementação do *software* [Kruchten, 1995].

Estas arquitecturas agrupam os componentes lógicos responsáveis pela resolução das necessidades do utilizador e demonstram os fluxos de informação e as

dependências entre os mesmos. Estas representações são independentes das linguagens ou paradigmas de programação a ser utilizados e não possuem informação sobre os endereços ou características dos dispositivos de *hardware* que lhes vão dar suporte.

***4 Step Rule Set* (4SRS)**. A técnica do *4 Step Rule Set* permite a geração de arquitecturas lógicas a partir dos requisitos elicitados ou de Casos de Uso [Fernandes et al., 2006; Machado et al., 2005]. As arquitecturas lógicas geradas são representadas usando a notação dos diagramas UML de objectos e componentes. Os diagramas de objectos, por sua vez, são representações mais alto nível das instâncias de um determinado sistema e as relações entre si, num determinado ponto temporal. Já os diagramas de componentes mostram a relação estrutural entre os componentes num determinado sistema. São usados maioritariamente na representação de plataformas complexas que podem ter muitos componentes diferentes. Os componentes trocam informação entre si através de *interfaces*.

O 4SRS, tal como o nome indica, é composto por quatro passos que permitem a transformação em componentes, perfazendo um diagrama de componentes [Fernandes & Machado, 2001]:

- 1º passo (*Component Criation*) – Transformar cada Caso de Uso em três objectos (um representativo de uma *interface*, um representativo de uma unidade de controlo e outro de armazenamento de dados);
- 2º passo (*Component Elimination)* – Partindo da descrição dos Casos de Uso, manter apenas os objectos gerados no 1º passo que, *per si*, tenham significado lógico para o sistema como um todo. Este passo, pela subjectividade que pode acarretar, pode ser dividido em sete micro passos [Azevedo et al., 2009];
- 3º passo (*Component Packaging and Aggregation)* – Dos objectos restantes no passo 2, verificar se pode existir lugar a operações de agregação ou associação dos objectos, tendo em conta que existem objectos que podem ter responsabilidades similares no contexto do sistema e que, por isso, podem ser agrupados;
- 4º passo (*Component Association)* – Os objectos agregados são conectados, especificando o tipo de ligação com os objectos existentes.

De seguida, ficam exemplos genéricos (e não contextualizados no âmbito deste trabalho) de um Diagrama de UML de Casos de Uso (Figura 3.1), que é o elemento inicial do processo de 4SRS e o consequente Diagrama UML de Componentes (Figura 3.2), resultado da aplicação da técnica acima descrita [Fernandes & Machado, 2001].

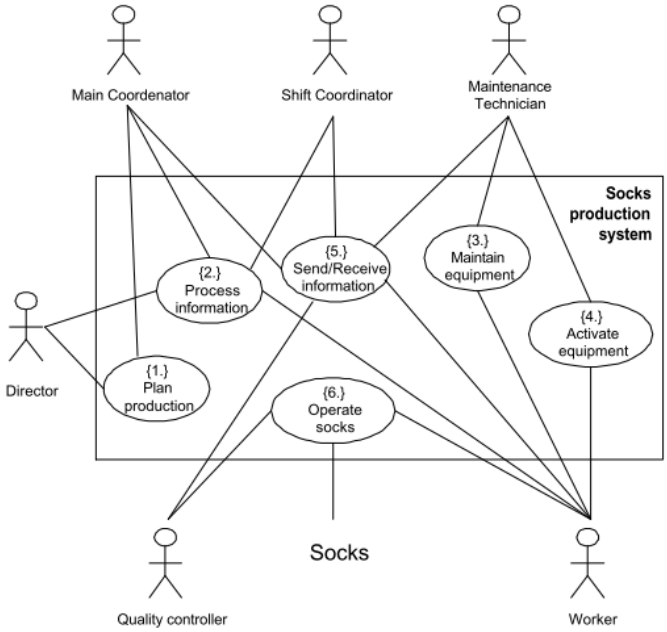

Figura 3.1 - Diagrama de Casos de Uso

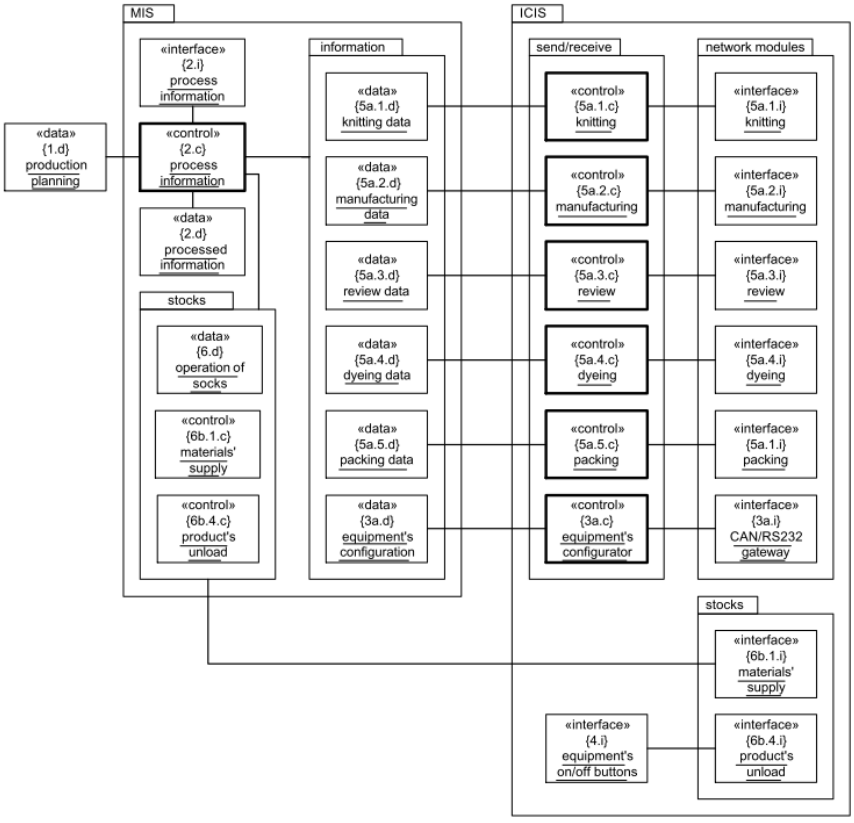

Figura 3.2 - Diagrama de Objectos gerado pela aplicação do 4SRS ao Diagrama da Figura 3.1

Recursividade e *Architectural Elements* (AEs). Por vezes, não é simples a percepção de quais os objectos que, *per si,* não incorporam relevância suficiente para figurarem como um AE na solução final. Em sistemas com dezenas ou centenas de Casos de Uso, o entendimento relativo aos objectos importantes pode advir de uma análise não determinística e que poderia levar a imprecisões com efeito no resultado do processo. Para contrariar este grau de incerteza, a técnica de 4SRS pode ser usada recursivamente de maneira a reduzir gradualmente a complexidade e o espectro de ambiguidade que pode advir deste processo. Adicionalmente, para possibilitar a automatização (ainda que parcial) do processo do 4SRS, foi gerado um modelo de tabela onde a informação relativa aos Casos de Uso e à aplicação dos sucessivos passos do processo pode ser armazenada, garantindo assim que a aplicação do 4SRS gera informação estruturada sempre da mesma maneira [Fernandes et al., 2006].

O 4SRS pode assim ser sintetizado como uma técnica de transformações recursivas à arquitectura do *software* de modo a conseguir representar os requisitos elicitados anteriormente. É uma técnica que mapeia os Casos de Uso (representação de requisitos e actores) em diagramas de componentes (representação das instâncias de um sistema) [Fernandes et al., 2006], composto por Elementos Arquitecturais (do inglês *Architectural Elements*), que são elementos que podem fazer parte de uma arquitectura lógica e que ficam relacionados com os artefactos gerados. Não existe uma descrição/representação estruturada deste tipo de elementos em *frameworks* como o UML, e a sua natureza pode variar consoante o tipo de sistema em estudo e o contexto onde são aplicados este tipo de elementos.

3.3 Processos de Geração de *User Stories* FLAUS

Partindo dos elementos arquitecturais que compõem os diagramas de objectos UML, as equipas de desenvolvimento conseguem ter uma percepção clara de como a implementação pode ser modularizada, quais os principais pontos críticos do sistema e qual o fluxo de informação e de acções existentes entre os diversos elementos. Apesar disso, não é fácil planear, dividir e prioritizar o trabalho de implementação, de modo a construir um *roadmap* partilhado pelos *stakeholders* e que permita a entrega às equipas de implementação de tarefas exequíveis e claramente delimitadas, que possam ser executadas em simultâneo e que não criem dependências temporais entre as mesmas.

Assim, e de carácter complementar à aplicação do processo do 4SRS, foi arquitecturado um outro processo que permitisse, além de ter uma representação clara do sistema através do diagrama de componentes resultante da aplicação do processo acima descrito, a divisão do trabalho de implementação em *User Stories*.

O procedimento, denominado de FLAUS, aproveita não só o resultado final da aplicação do 4SRS, como da documentação auxiliar gerada pelos diversos passos do processo, e que servem inicialmente para ajudar a descrever cada um dos elementos arquitecturais gerados.

O objectivo final do FLAUS passa então pela geração de *User Stories*, que respeitem as características INVEST apresentadas no capítulo anterior, e que possam ser directamente inseridas num *Backlog* de uma equipa existente e que tenha o seu próprio processo de desenvolvimento ágil de *software*.

***Product-Level Logical Architecture Diagram*.** O ponto de partida para a execução e criação deste processo é o resultado final do processo de 4SRS – o diagrama de arquitectura lógica do produto. Considerando produtos muito complexos, é fácil de perceber que estes modelos podem ser extremamente extensos e pesados para serem analisados como um todo, pois nestes diagramas estão representados todos os módulos necessários para executar todos os requisitos funcionais pretendidos. Além do mais, não é expectável que, para sistemas complexos, seja apenas uma equipa de implementação a executar todo o trabalho. Num modelo onde podem existir centenas de módulos, uma equipa de implementação que siga a abordagem metodológica do *Scrum* (que por definição, não deve ter mais de 10 elementos) poderia demorar vários anos até conseguir implementar todos os detalhes de modelos complexos. Esta quantidade de tempo não é exequível com as necessidades de um mercado dinâmico, onde existe sempre grande urgência de lançar produtos no mercado para ir de encontro às necessidades levantadas previamente.

Assim, uma das preocupações principais do FLAUS é o agrupamento das *User Stories* por categorias, de maneira a existir um agregamento lógico das unidades de trabalho que permita a existência de múltiplas equipas de implementação a trabalhar em paralelo, diminuindo consideravelmente o tempo necessário para implementar e entregar a solução ao cliente.

Introdução ao processo FLAUS. A primeira grande questão na criação deste processo foi perceber qual seria a relação directa entre os AEs e as *User Stories* geradas. Na aplicação do processo do 4SRS, os AEs são gerados através da decomposição de Casos de Uso em três contextos diferentes. Faria então sentido criar uma *User Story* para cada Caso de Uso ou era mais oportuno mapear directamente cada AE numa *User Story*? Mantendo os princípios fundamentais para a escrita de *User Stories* (características INVEST), foi tomada a opção de criar uma *User Story* para cada AE.

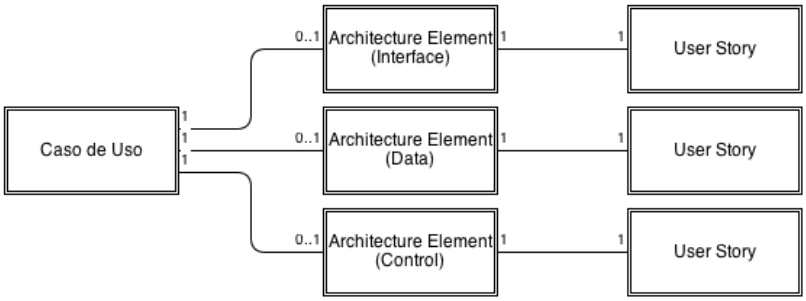

Figura 3.3 – Relação entre Casos de Uso, AEs e *User Stories*

Foram vários os motivos que levaram a esta decisão, nomeadamente a maior flexibilidade conferida ao *Product Owner* para acompanhar o trabalho de equipa. Se as *User Stories* forem sempre complexas (em certos casos poderiam ter um contexto de acesso aos dados, outro contexto de comunicação e ainda outro contexto de controlo) e requererem grande esforço de implementação, corre-se o risco de diluir uma das principais vantagens reconhecidas às metodologias ágeis: a facilidade de mudar o rumo da equipa e a capacidade de se perceber, em tempo real, qual o estado da equipa em relação ao compromisso para uma determinada *Sprint.* Quando se implementa *User Stories* de grande complexidade, que podem ocupar a *Sprint* inteira, apenas se pode tirar conclusões sobre a velocidade e o empenho da equipa no final da *Sprint,* quando o trabalho (supostamente) deve estar terminado. Estes argumentos vão de encontro às características INVEST de obter *User Stories* pequenas (*Small*) e, por via desta característica, simplificar a estimativa de esforço de implementação (*Estimable*).

Documentação. A aplicação do 4SRS sobre os Casos de Uso elicitados, origina bastante documentação e informação, que serviu de auxílio para a execução dos vários passos do processo. Da documentação, destacam-se as informações relativas à especificação dos Casos de Uso e às especificações e descrições dos AE gerados. Normalmente esta documentação é criada no passo 2 para ajudar na sua execução e na

decisão de quais os AE a eliminar, servindo também como documentação auxiliar que permite entender detalhadamente o escopo de cada um dos AEs da solução final. Esta documentação, juntamente com os actores associados a cada Caso de Uso (de onde descendeu cada AE) são elementos fundamentais na geração das *User Stories*, dado que neles estão encapsuladas informações necessárias para a escrita de *User Stories* que respeitem os princípios INVEST.

Para a geração de *User Stories,* partindo dos elementos da arquitectura lógica resultante do 4SRS, é sugerido um processo com 3 passos, que serão descritos e exemplificados de seguida. Para ajudar na compreensão do processo FLAUS, foi criado um diagrama de Casos de Uso e um diagrama lógico de componentes, ambos descontextualizados, que serve apenas para o propósito de exemplo nesta dissertação (Figura 3.4).

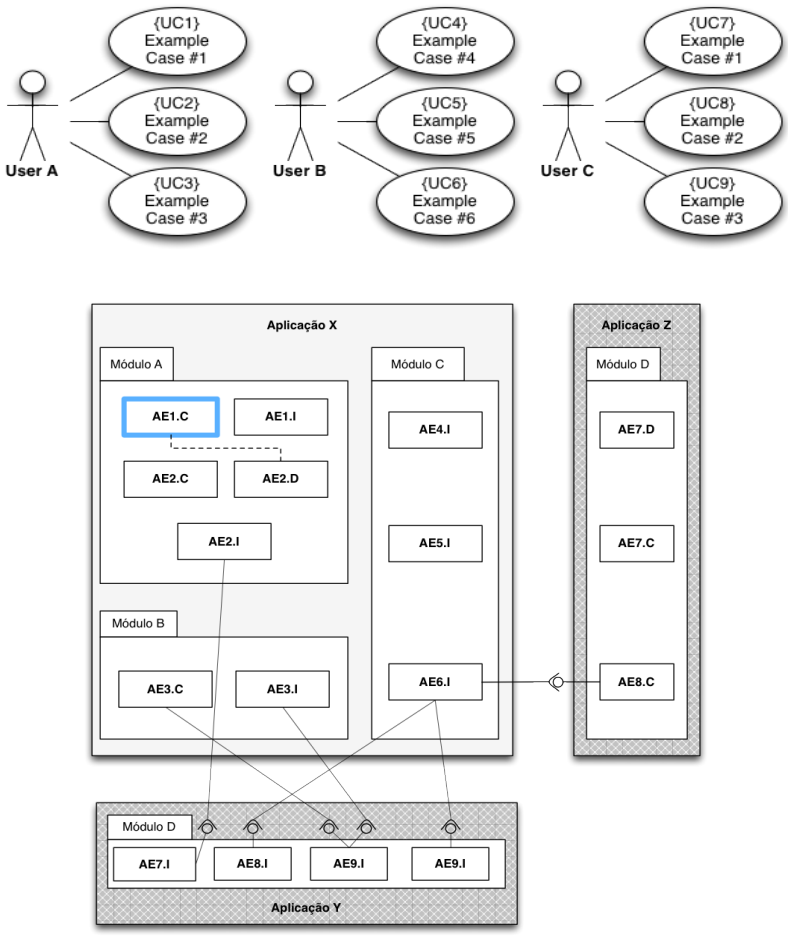

Figura 3.4 - Exemplo de diagrama de Casos de Uso e respectiva Arquitectura Lógica gerada por eventual aplicação do 4SRS

1º Passo – Listagem de AEs. O primeiro passo consiste em identificar e isolar os AEs respeitantes a cada aplicação, de modo a manter o agrupamento lógico definido

pela aplicação do 4SRS. Adicionalmente, é necessário analisar o diagrama de arquitecturas lógicas e perceber as ligações entre os AEs, listando-os e retirando algumas informações como o seu tipo (*interface, data* ou *control).* Para cada aplicação faz-se a listagem não apenas dos AEs que a compõem mas também daqueles que têm ligação directa com qualquer um dos AEs integrantes (mesmo que pertencentes a outras aplicações).

Desta análise, é possível gerar duas tabelas com a seguinte estrutura:

- Tipo – Se o AE é do tipo *control, interface* ou *data;*
- Número – Contagem do número de AEs;
- Lista – Listagem dos AEs em questão.

AE componentes				
Tipo	Número	Lista		
Control	3	1.C	2.C	3.C
Data	1		2.D	
Interface	6	1.I	2.I	3.I
		4.I	5.I	6.I

Tabela 3.1 - Lista exemplo de AEs referentes ao aplicação X

AE de ligação			
Módulo	Tipo	Número	Lista
Módulo Y	*Interface*	4	7.I 8.I 9.I 10.I
Módulo Z	*Data*	0	-
	Control	1	8.C

Tabela 3.2 - Lista exemplo de AEs de ligação à aplicação X

Analisando estas duas tabelas, além de uma perspectiva geral sobre o principal foco da aplicação a desenvolver (se é ligado para acções relacionados com manipulação e armazenamento de dados, acções relacionados com a troca de informação entre componentes ou acções relacionados com operações lógicas sobre a informação), tem-se também uma perspectiva sobre que outras aplicações estão mais directamente

relacionados com aquele que se vai implementar, o que pode permitir uma política de maior cooperação e proximidade entre as equipas responsáveis pela sua implementação.

2º Passo – Preenchimento da *User Story Card*. O próximo passo é analisar o "rasto" deixado pela aplicação do 4SRS que permitiu a criação do AE. As informações recolhidas, embora não sejam todas necessárias para a geração do título da *User Story*, são fundamentais para que todos os profissionais envolvidos na sua implementação percebam claramente o que é pretendido, para quem é pretendido, porque é que é pretendido e ainda com quem é pretendido. Estas informações provêm dos diversos passos preconizados pelo 4SRS para gerar o AE. Todos estes detalhes de cada AE devem ser armazenados com a mesma estrutura para darem lugar à criação de uma "ficha" para cada *User Story*, contendo toda a informação necessária para proceder à sua estimação e posterior implementação. Deste modo, para cada AE é importante obter as seguintes informações:

- Nome – Nome do AE, retirado directamente do diagrama lógico
- Código – Código identificativo do AE
- Tipo – Se é do tipo *interface / data* ou *control*
- Descrição – Descrição do AE, normalmente presente na documentação auxiliar da aplicação do 4SRS
- Múltiplo – Quando o AE pertence a mais do que uma aplicação diferente.
- Módulo – Nome do módulo onde está incluído o AE no diagrama lógico
- Ligações
    - Associação Directa / Ligação por Caso de Uso – AEs directamente associados através da execução do 4º passo do 4SRS
- Caso de Uso Associado
    - Actores Envolvidos – Actor envolvido no Caso de Uso que deu origem ao AE
    - Caso de Uso Ascendente – Caso de Uso de nível superior do Caso de Uso que deu origem ao AE.

Paralelamente a estas informações, é igualmente importante saber se o AE em questão está presente em mais do que uma aplicação (existem casos onde AE são constituintes directos de 2 e 3 aplicações simultaneamente).

Para manter a compatibilidade com as equipas que têm hábitos de manter a informação das *User Stories* sempre visível (colocando as suas características em formato físico, muitas vezes em forma de cartões) foi criado um *template* para um formulário da *User Story*, denominado *User Story Card,* que contempla todas as informações recolhidas e listadas anteriormente, assim como algumas informações que a equipa de implementação vai gerar no *Grooming*, como o número de *Story Points,* os critérios de aceitação, ou algum outro comentário que a equipa ache pertinente registar e guardar.

user story	US #	(nome)**		
	Critérios Aceitação	*		*Story Points* *
architectural element	AE #		(nome)	
	Tipo		Módulo	
	Descrição			
	Múltiplo		Módulo	
	Ligação	Assoc.		
		Ind.		
caso de uso	CU #		(nome)	
	Descrição			
	CU Asc.			
	Actores			
(comentários)				

Tabela 3.3 - *Template* de *User Story Card*

(* a preencher pela equipa de implementação)
(** apenas disponível após a execução do 3º passo)

Como se pode verificar por análise à Tabela 3.3, para qualquer *stakeholder* (desde o cliente/*Product Owner* até á equipa de implementação) existe neste cartão informação útil, cuja leitura é directa e simples. O preenchimento do referido formulário pretende também ser básico e rápido pois toda a informação referente ao AE e ao Caso de Uso está disponível pela utilização do 4SRS, enquanto que a informação referente à *User Story (Story Points* e critérios de aceitação) é obrigatória e é definida pela equipa de implementação durante o *Grooming*. De seguida, pode-se observar um exemplo da

utilização do referido modelo, utilizando para o caso o AE.1.c que seria hipoteticamente gerado a partir da execução do 4SRS ao "*Example Case*" #1.

user story	US #			
	Critérios Aceitação	*		*Story Points*
				*
architectural element	AE #	1.C	AE 1.C	
	Tipo	*Control*	Módulo	Package A
	Descrição	N/D		
	Múltiplo	NÃO	Módulo	N/A
	Ligação	Assoc.	AE 2.D	
		Ind.	x	
caso de uso	CU #	1	Example Case #1	
	Descrição	N/D		
	CU Asc.	N/A		
	Actores	User A		
x				

Tabela 3.4 - Aplicação exemplificativa do 2º passo do processo FLAUS no AE 1.C

3º Passo – Redigir *User Story*. O último passo, aquele que vai efectivar e gerar o resultado final de todo o processo. Uma das grandes vantagens da aplicação do 4SRS na geração de AE, é que rapidamente se percebe qual vai ser o seu objectivo final: manipulação de dados, comunicação ou operações lógicas, através da leitura do tipo de AE. Esta estandardização dos tipos de AE permite simplificar a gestão de *User Stories*, pois, ao fim ao cabo, estão centradas em três tipos muito específicos de tarefas.

Relembrando a composição de um título da *User Story*, existem três partes fundamentais para a escrita de uma boa *User Story*: "quem" (*who*), "o quê" (*what*) e o "porquê" (*why*):

> *"As a <role>, I need/want to have a(n) <what-description>, in order to <why-outcome>."*

O "quem", os actores envolvidos e que vão executar tarefas sobre a *User Story* a implementar, são facilmente identificados analisando o respectivo *User Story Card* e procurando os actores envolvidos no Caso de Uso que gera o AE pela aplicação do 4SRS. Como a lógica envolvente à necessidade de representar propriedades dos sistemas em Casos de Uso e *User Stories* é semelhante (capturar requisitos específicos em termos de interacção entre utilizadores e sistema), é fácil validar que os actores envolvidos nos Casos de Uso vão ser os actores beneficiados pela implementação de uma determinada *User Story*.

Relativamente ao "o quê", também é possível encontrar um relacionamento directo entre esse componente e o nome do AE. Inicialmente é necessário encontrar uma acção, representada por um verbo, que permita identificar o que se quer implementar. A divisão entre AEs de *control, interface* e *data* simplifica essa procura, dado que os AEs de *interface* referem-se sempre à criação de uma determinada *interface*, sendo por isso uma acção fixa e constante que é a necessidade da existência de um canal para comunicação entre componente e/ou actores. Assim, nos casos dos AEs de *interface*, os actores envolvidos apenas necessitam que os mesmos existam para conseguir utilizá-los no seu fluxo de acções. Usando o nome do AE correspondente, e utilizando as conexões de "*want/need to have"* (quero/necessito de ter), está representada a ligação entre o "quem" (actor) e quê (acção).

Nos casos dos AEs de *interface* que não tenham este nome, a parte central da *User Story* relativamente ao "o quê" é deixada simplesmente com a informação *want/need to have an interface* (preciso/necessito de ter uma *interface*), sendo transportado o título do AE (acções que se vão desenrolar utilizando aquela *interface)*, para a parte do "porquê".

Os AEs do tipo *data* possuem uma uniformidade semelhante, dado que se referem normalmente à necessidade de existência de repositórios/locais de armazenamento ou ainda de interfaces para comunicação com esses referidos espaços de armazenamento. Deste modo, a construção da *User Story* segue a mesma regra utilizada nos AEs de *interface*.

Comparativamente com os dois tipos anteriores, os AEs de controlo podem ser bastante díspares entre si. Eles suportam a lógica por detrás de um sistema,

representando todas as acções que se podem executar manipulando os dados (representados pelos AEs de *data)* e utilizando *interfaces* para a sua transmissão (representados pelos AEs de *interface*). Como podem representar qualquer tipo de acção no sistema, os AEs de controlo costumam ter associado um verbo que representa a acção que é necessário executar. Neste caso, como se está a transportar informação de um título para uma frase, pode ser necessário algum tipo de correcção semântica das palavras para que a frase possa fazer sentido (capítulo 4.3).

Por fim, falta especificar o "porquê" de um determinado actor ("quem") poder necessitar de executar uma determinada acção ("o quê"). Esta informação, embora muitas vezes possa ser induzida através do próprio título do AE correspondente, pode ser complementada com a descrição do Caso de Uso que lhe deu origem. Como as *User Stories* acabam por estar a um nível mais detalhado e baixo-nível que um Caso de Uso, a própria descrição do Caso de Uso consegue justificar a necessidade de existência de uma determinada *User Story*. Nos casos em que o nome do AE é bastante semelhante ao do Use Case que o originou, deve-se encontrar na descrição do Caso de Uso outros termos que ajudem a explicar a relevância da *User Story*.

3.4 Limitações deste Processo

Apesar de não ter sido aplicado mais do que uma vez num projecto "real", é possível observar pela leitura atenta dos procedimentos que compõem o processo FLAUS, que este tem limitações e que precisa de testes mais aprofundados.

A primeira limitação intrínseca à utilização do processo FLAUS é que a geração das *User Stories* a partir de um diagrama lógico, mesmo que seja representativo de todas as funcionalidades do sistema, não garante que cubra todos os detalhes necessários à implementação de um produto completo. As tarefas relacionadas com outros contextos que não a implementação das funcionalidades, como o cumprimento dos requisitos não funcionais, não estão contemplados no resultado final do processo FLAUS. Isto pode levar a uma desfocalização no cumprimento deste tipo de detalhes, o que pode ter um impacto negativo na qualidade do produto final.

Outro ponto negativo prende-se com a dependência da aplicação do 4SRS no seu modo mais profundo, com a geração de toda a documentação auxiliar necessária para o processo FLAUS. Embora o ponto inicial para a aplicação do FLAUS seja um "vulgar"

diagrama lógico de um sistema em UML, apenas se o mesmo for gerado recorrendo ao 4SRS é que o processo descrito está apto a ser aplicado.

Tal como foi referido anteriormente, existe mais um ponto fraco no processo que dificulta a sua aplicação de um modo automático. Confiando na semântica dos AEs e dos Use Cases que deram origem ao diagrama lógico, podem existir casos onde a aplicação rígida do 3º passo possa originar *User Stories* com pouca coerência semântica, repetindo ideias (caso o AE e o Caso de Uso que lhe deu origem tenham nomes semelhantes) ou não existindo todos os elementos necessários para formar uma frase válida (substantivo, verbo e complementos), obrigando a uma correcção manual da linguagem.

3.5 Conclusão

Apesar do *Scrum* não referenciar a necessidade da introdução e especificação de documentação formal, o projecto é melhor controlado e as funcionalidades mais facilmente implementadas quanto maior for a informação disponibilizada aos intervenientes no processo. A comunidade ágil na sua maioria considera as técnicas de engenharia de requisitos dos modelos *Waterfall* incompatível com os princípios ágeis, devido à sua necessidade de documentação extensa. Apesar disso, existem já esforços para começar a adaptar algumas técnicas dos modelos mais formais de modo a conseguir introduzir melhorias no capítulo da engenharia dos requisitos e na modelação de sistemas nas metodologias ágeis [Paetsch et al., 2003].

Seguindo essa nova corrente, a existência de diagramas lógicos do produto deve ser encarado como um ponto positivo em todo o processo de implementação. A sua conjunção com os métodos ágeis, tradicionalmente despegados de documentação formal, deve ser vista como uma oportunidade para enriquecer e melhorar a eficiência das equipas de implementação.

O processo FLAUS, além de gerar de um modo semi-estruturado as *User Stories,* adaptando-as e enriquecendo a sua descrição com informação mais formal e completa, consegue conjugar-se com diagramas lógicos do produto, documentação e informação que as equipas *Scrum* normalmente não têm acesso.

4. O Projecto ISOFIN em User Stories

4.1 Introdução

Como já foi referido anteriormente, no panorama das empresas (pequenas e médias) portuguesas que operam no ramo da implementação de *software*, um dos principais focos de rendimento são os concursos públicos. Sejam nacionais ou internacionais, com fundos portugueses ou ajudas comunitárias, existem diversos projectos onde são entregues os cadernos de encargos e é acordada a adjudicação do projecto e, muitas vezes, o contacto seguinte com o cliente final coincide com a altura da instalação do produto. Apesar desse afastamento do cliente final, as equipas continuam a utilizar as metodologias ágeis e descuram muitas vezes as fases de especificação e arquitectura da solução. Para dar resposta a essa menor capacidade existente, propõe-se a utilização dos 4SRS para criar uma arquitectura lógica representativa do sistema e o processo FLAUS, capaz de gerar *User Stories* que possam ser correctamente interpretadas pelas equipas que seguem abordagens metodológicas como o *Scrum*.

Neste capítulo, vai ser apresentado um projecto co-financiado por fundos do Quadro de Referência Estratégica Nacional (QREN), no qual a Universidade do Minho participou com o seu conhecimento na especificação dos requisitos e criação de arquitecturas para o sistema. É sobre essa arquitectura que vai ser aplicado o processo FLAUS, capaz de gerar *User Stories* (agrupadas em aplicações) e que podem ser distribuídas às várias entidades do consórcio que são responsáveis pela implementação do produto.

ISOFIN. Um cliente (que pode ser alguém do público geral, ou até uma das entidades vistas como fornecedoras de serviços financeiros) pode necessitar de um produto que seja constituído por múltiplos serviços financeiros, prestados por entidades diferentes. Desta forma, aplicações financeiras complexas podem ser formadas, a informação sobre clientes partilhada, o que permite uma resposta mais eficaz às necessidades dos clientes.

O principal objectivo do projecto ISOFIN é o desenvolvimento da plataforma ISOFIN *Cloud*, que visa a unificação ontológica do domínio da banca e seguros, de forma a que clientes, *brokers*, banca e seguradoras possam aceder de forma transparente aos serviços existentes na *Cloud*. A construção da Plataforma ISOFIN *Cloud* pressupõe a definição de uma metodologia e a construção de ferramentas que facilitem o desenvolvimento e disponibilização dos serviços lá criados. Com base nestas ferramentas que se encontram embutidas na plataforma, serão desenvolvidos serviços/aplicações para o domínio financeiro, banca e seguros que serão integrados na *Cloud*.

No início do projecto, embora se conhecessem as finalidades da plataforma, não existiam quaisquer requisitos formalizados, sejam eles de índole funcional ou não funcional. Foi trabalho de uma equipa da Universidade do Minho, em união de esforços com o Centro de Computação Gráfica, a elicitação de requisitos funcionais para o sistema, assim como o a modelação da arquitectura lógica. Como a arquitectura processual do sistema não estava definida, foi necessária a criação de diagramas UML de sequência de maneira a perceber como as diversas componentes deveriam interagir, de modo a realizar as tarefas necessárias.

O objectivo final desta equipa seria produzir um diagrama de componentes lógicas que facilitasse a tarefa da divisão e implementação do trabalho por parte das equipas que irão implementar o *software*. Uma das técnicas escolhidas, devido à sua fácil interpretação pelos programadores, foi os diagramas de arquitecturas lógicas baseadas em componentes UML, e que são o resultado final do processo do 4SRS.

Foi a partir dessa arquitectura lógica (e de toda a documentação auxiliar gerada entretanto) que se aplicou o processo FLAUS, descrito no capítulo anterior.

4.2 Geração de *User Stories* através do processo FLAUS

Com o término da representação do sistema ISOFIN a implementar, e fruto da necessidade das equipas de desenvolvimento em receber o esforço de implementação especificado de modo que possa ser interpretado por uma equipa *Scrum*, foi criado o processo FLAUS, que pudesse transformar as arquitecturas em *User Stories* que respeitem as características INVEST apresentadas no capítulo anterior.

***ISOFIN Product-Level Logical Architecture Diagram*.** O ponto de partida para a execução e criação deste processo é o resultado final do processo de 4SRS – o diagrama de arquitectura lógica do produto.

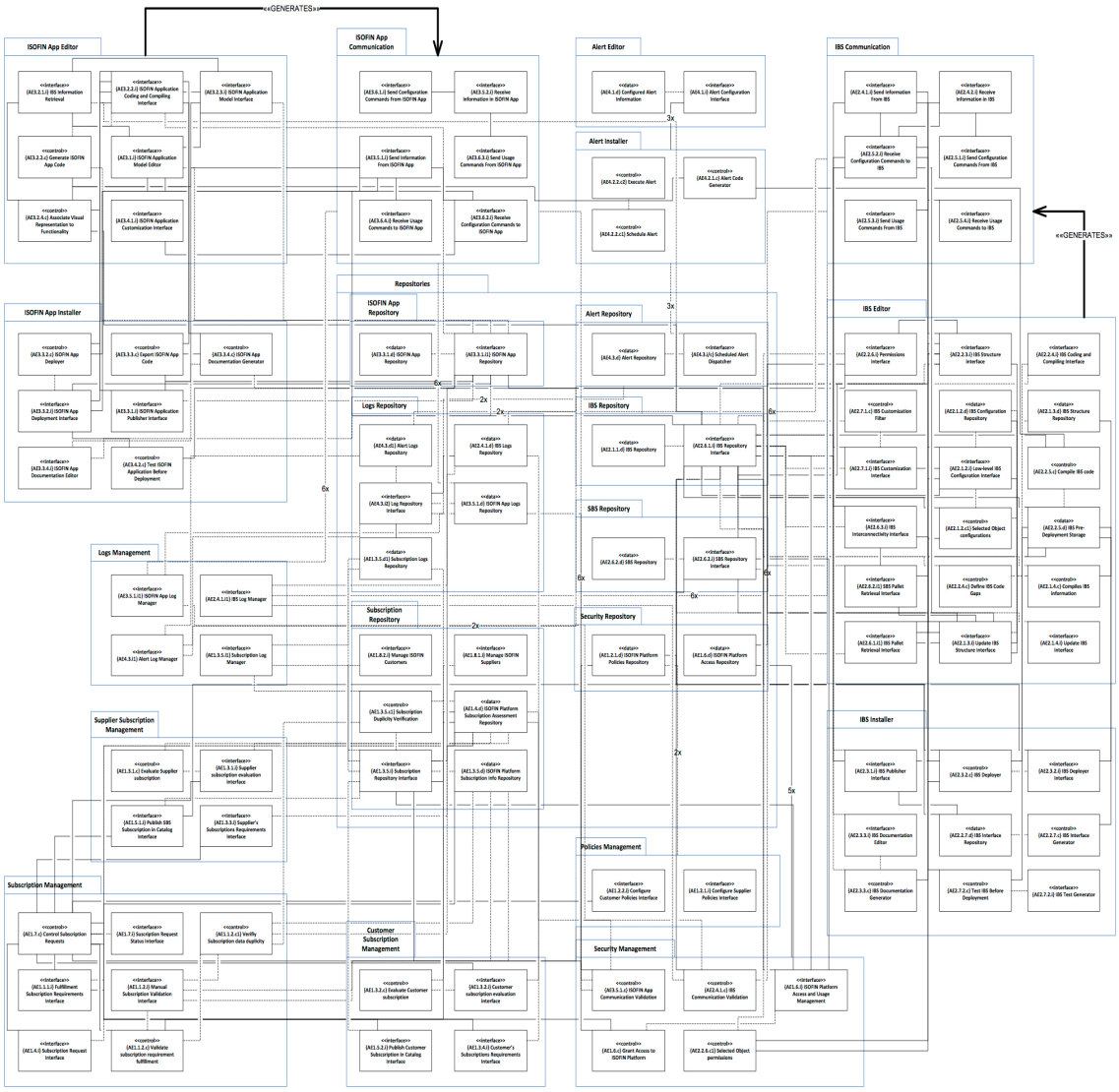

Figura 4.1 - Arquitectura lógica do produto ISOFIN

No caso concreto do ISOFIN, o diagrama da arquitectura lógica era complexo e difícil de analisar como um todo. Contém 113 AEs, divididos em 21 módulos diferentes. Na Figura 4.1 pode-se ver como o diagrama lógico do ISOFIN é extenso e complexo, e como é mais simples analisar o sistema a um nível mais abstracto, verificando o mesmo diagrama colapsado através de um agrupamento dos AEs em módulos (Figura 4.2).

Apesar dos AEs estarem agrupados no diagrama por módulos, não foi essa a lógica utilizada para fazer a divisão do esforço de implementação por equipas. Numa primeira análise, poderia fazer algum sentido entregar a cada uma delas a responsabilidade de implementar um componente que tivesse um contexto pré-definido e único. As equipas poderiam assim especializar-se num determinado contexto e utilizar essa especialização para construírem uma implementação mais próxima ao contexto real. No entanto, a lógica principal por detrás da necessidade em erguer esta plataforma é exactamente a união, num único espaço, de todas as realidades presentes no universo financeiro e de seguros. Assim, foi decidido dividir a plataforma por áreas de acção, tentando aproximar as áreas aos principais domínios da plataforma.

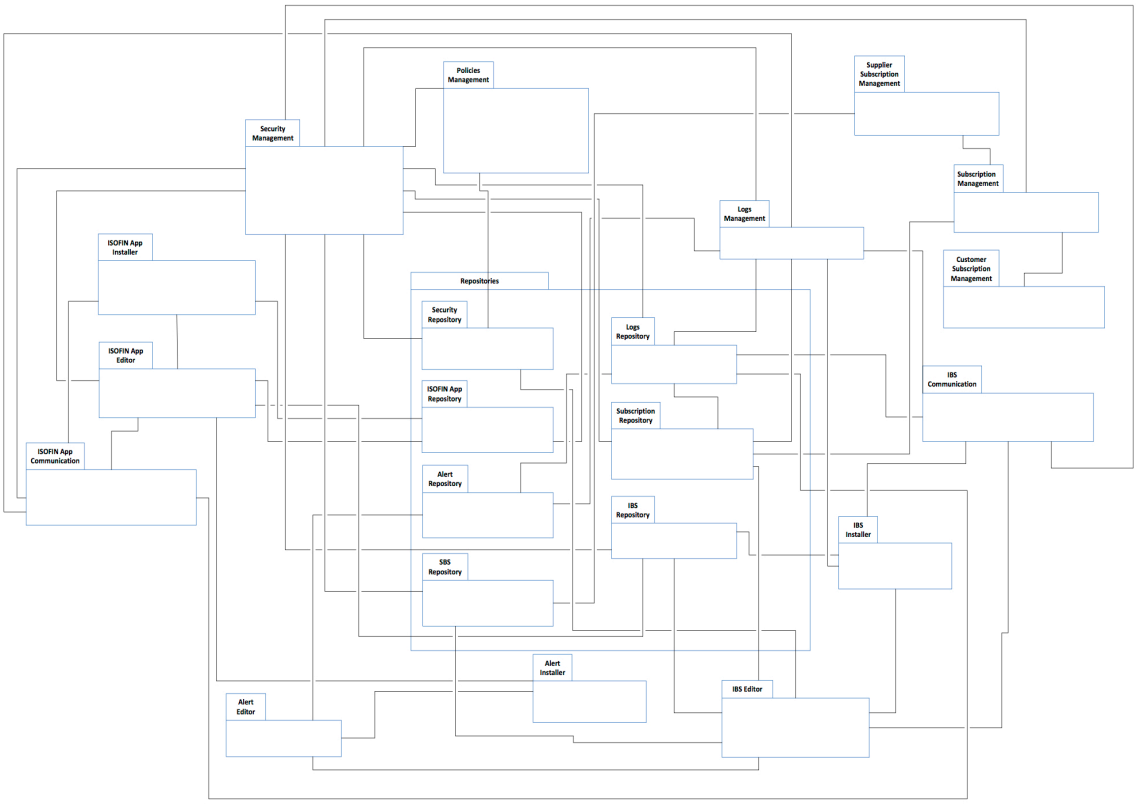

Figura 4.2 - Arquitectura lógica do produto ISOFIN colapsada por módulos

Foram identificados 7 áreas de acção (aplicações) na plataforma ISOFIN (Figura 4.3):

- ISOFIN *Application Management*
- IBS *Management*
- *Alert Management*
- *Subscription Management*
- *Security Management*
- *Policies Management*
- *Logs Management*

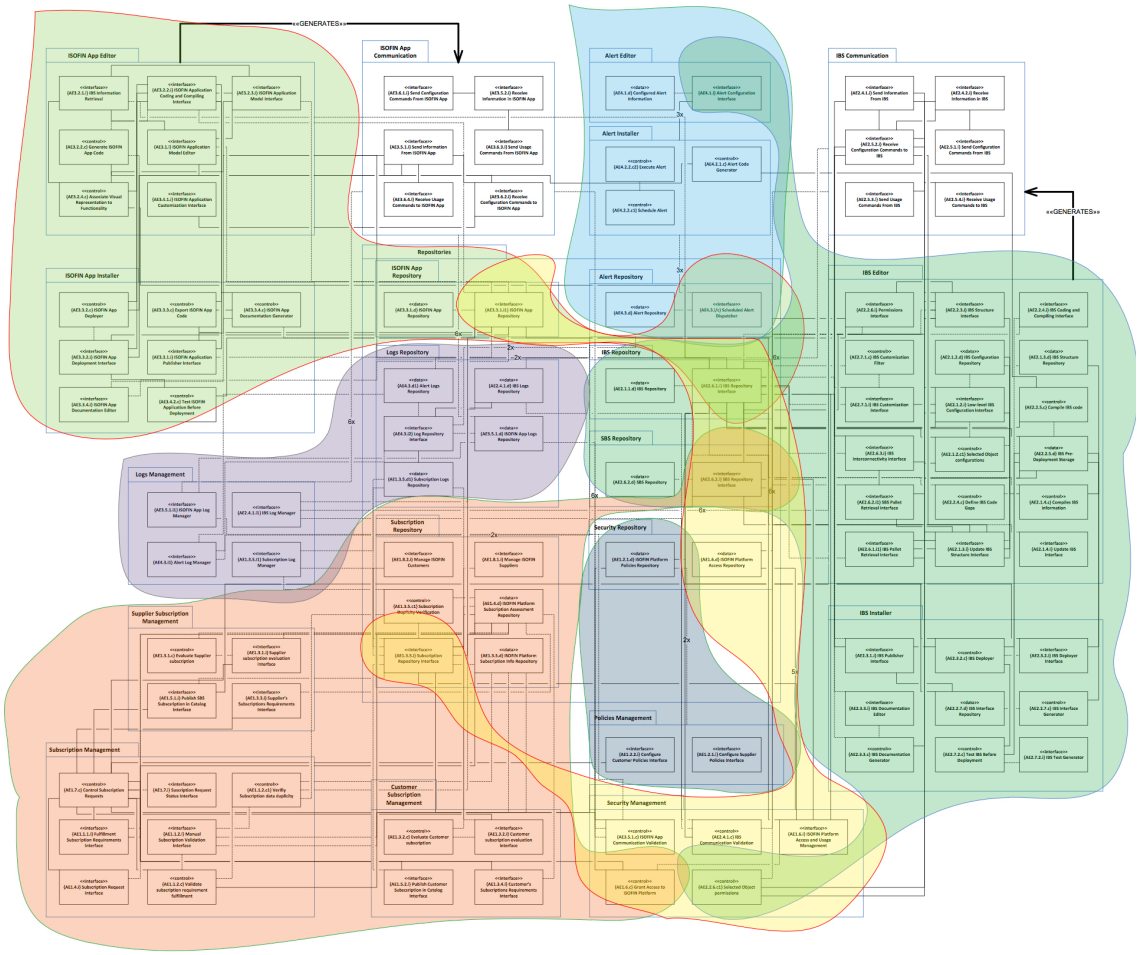

Figura 4.3 – Arquitectura lógica do produto ISOFIN agrupada por aplicações

Convém referenciar que a lógica utilizada pelas equipas de execução do 4SRS para dividir as AEs nestas áreas foi também a utilizada para agrupar as *User Stories* geradas em categorias diferentes. No âmbito deste trabalho, foi escolhida uma área para executar e validar o processo de transformação de diagramas de arquitecturas lógicas.

Foi escolhida a área de *IBS Management* (Gestão de IBS) por ser uma área representativa da plataforma, que permitia executar a acção fundamental da plataforma: o utilizador pode criar o seu próprio IBS. Um IBS (*Interconnected Business Service)* é um conjunto de funcionalidades disponibilizadas pela plataforma ISOFIN para os seus clientes. É constituída por um conjunto de outros IBSs e/ou SBSs (produtos existentes por parte dos parceiros da rede, disponibilizados na plataforma) e mapeiam directamente a necessidade de um determinado utilizador.

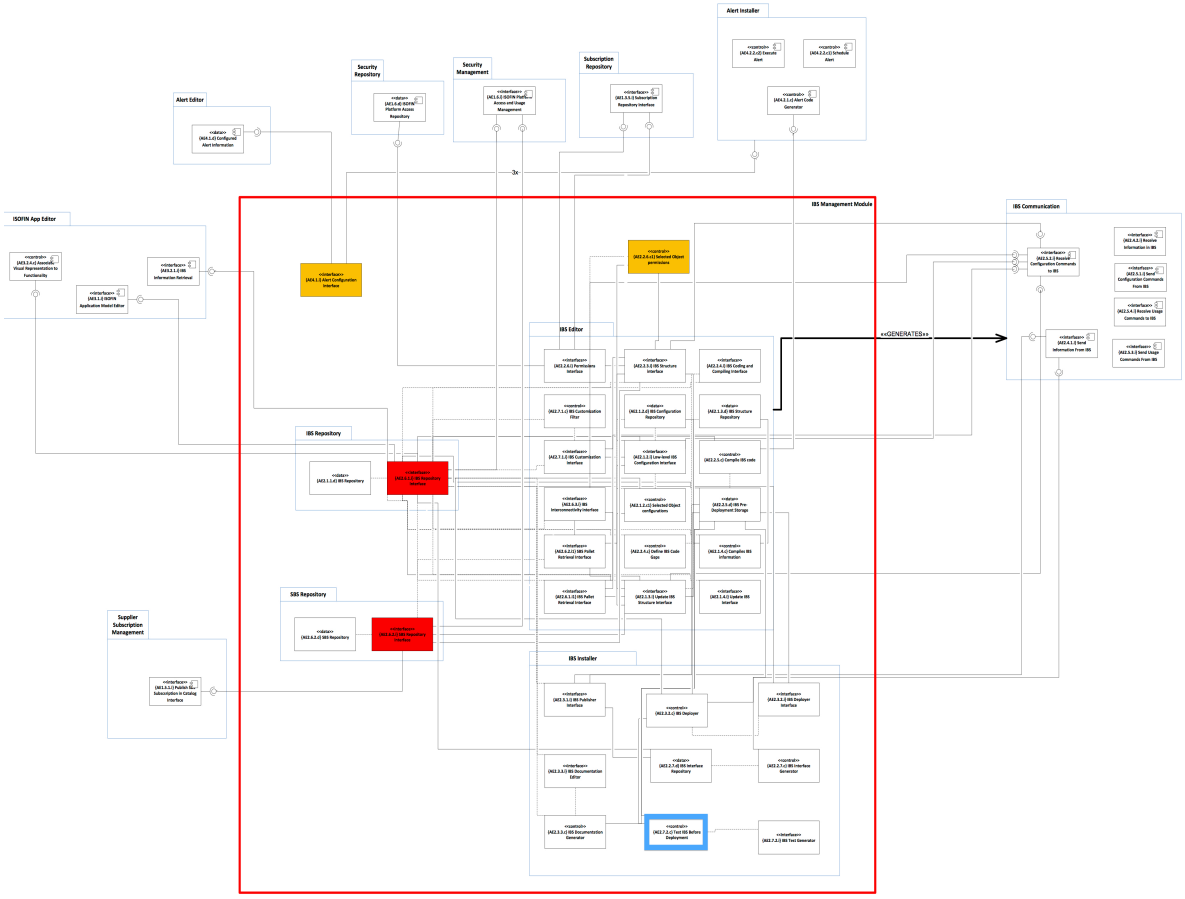

Figura 4.4 - Vista geral da aplicação *IBS Management*

Além de ser representativa, esta área (Figura 4.4) possui AEs dos três tipos: *data, interface* e controlo e continha AEs pertencentes a quatro módulos diferentes (*IBS Repository, SBS Repository, IBS Editor* e *IBS Installer*). Adicionalmente, esta área possui dois AEs (AE 4.1.i – *ISOFIN Application Customization Interface* e o AE 2.2.6.c1 – *Selected Object permissions*) que são originários de módulos pertencentes a outras aplicações que não o *IBS Management*. A amarelo estão representados então os AEs que estão incluídos em duas aplicações diferentes. A vermelho são os AEs que estão incluídos na lógica de três aplicações diferentes. Esta sobreposição de AEs pode originar

eventuais pontos críticos na implementação, dado que o mesmo AE está presente na solução a ser executada de até três equipas diferentes.

Este conjunto de factores resulta numa área representativa de todas as particularidades de um diagrama de arquitectura lógica, o que ajuda a validar os resultados obtidos pela aplicação deste processo.

De seguida, vai ser aplicado o processo FLAUS ao diagrama lógico da aplicação *IBS Management* (Figura 4.4), de maneira a ser gerado, no final do processo, um conjunto de *User Stories* que representem todos os requisitos funcionais que implementem todas as funcionalidades da aplicação escolhida.

1º Passo – Listagem de AEs. O primeiro passo, tal como foi especificado em cima, é o mais simples e o que não oferece grandes dificuldades, seja qual for o escopo em questão. Ainda assim, a sua execução permite logo perceber a predominância do tipo de acções que esse aplicação vai executar no seio do sistema a implementar.

AE componentes				
Tipo	Número	Lista		
Control	10	2.1.2.c1	2.1.4.c	2.2.4.c
		2.2.5.c	2.2.6.c1	2.2.7.c
		2.3.2.c	2.3.3.c	2.7.1.c
		2.7.2.c		
Data	6	2.1.1.d	2.1.2.d	2.1.3.d
		2.2.5.d	2.2.7.d	2.6.2.d
Interface	16	2.1.2.i	2.1.3.i	2.1.4.i
		2.2.3.i	2.2.4.i	2.2.6.i
		2.3.1.i	2.3.2.i	2.3.3.i
		2.6.1.i1	2.6.1.i	2.6.2.i1
		2.6.2.i	2.6.3.i	2.7.1.i
		4.1.i		

Tabela 4.1 – Lista de AEs componentes do módulo *"IBS Management"*

A diferença entre as duas tabelas também é facilmente perceptível pela observação da Figura 4.4: A Tabela 4.1 é referente aos AEs que a equipa responsável pela aplicação *IBS Management* vai ter de implementar, delimitados pelo quadrado vermelho da figura. A Tabela 4.2 é referente a todos os AEs, não incluídos no quadrado vermelho, mas que comunicam com componentes internos da aplicação.

Os AEs 2.2.6.c1 e 4.i.1, assinalados com asterisco na tabela anterior, são partilhados por mais do que uma aplicação e o restante módulo a que pertencem não faz parte da aplicação *IBS Management.* Assim, embora apareçam na Tabela 4.1, apenas lhe vão ser aplicados os passos seguintes do processo FLAUS quando as aplicações que integram os módulos a que pertencem sofrerem esse processo.

AE de ligação			
Módulo	Tipo	Número	Lista
Subscription Repository	*Interface*	1	1.3.5.i
Supplier Subscription Management	*Interface*	1	1.5.1.i
Logs Management	*Interface*	6	2.4.1.i 2.4.2.i 2.5.1.i 2.5.2.i 2.5.3.i 2.5.4.i
Security Repository	*Data*	1	1.6.d
Security Management	*Interface*	1	1.6.i
ISOFIN App Editor	*Control*	1	3.2.4.c
	Interface	2	3.1.i 3.2.i
Alert Editor	*Data*	1	4.1.d
Alert Installer	*Control*	3	4.2.1.c 4.2.2.c1 4.2.2.c2

Tabela 4.2 – Lista de AEs com ligação directa ao módulo "IBS Management"

Da análise à tabela anterior, facilmente se depreende que vai ter de existir uma sincronização cuidada com a(s) equipa(s) responsável(eis) pelas aplicações a que pertencem os módulos *Logs Management*, *ISOFIN App Editor* e *Alert Installer*, devido aos pontos de comunicação com os AEs da aplicação *IBS Management*.

2º Passo – Preenchimento da User Story Card. De seguida, pode-se observar um exemplo mais concreto do preenchimento da *User Story Card* (Tabela 3.3), artefacto que agrega todas as informações relativas a uma *User Story* e aos AEs / Casos de Uso que estão na sua origem, de modo a que a equipa de implementação tenha reunido toda a informação relativamente à funcionalidade a implementar.

Para o efeito, foi escolhido um AE da aplicação modelada no Passo 1, preenchendo-se o *User Story Card*. As informações relativas à primeira parte da tabela (*User Story*), só são preenchidas pela própria equipa de implementação na altura do *Grooming*. O nome da *User Story* só é gerado no passo 3. O AE que foi escolhido para este exemplo é o AE 2.7.2.c com a designação "*Test IBS Before Deployment*". Este AE pode ser localizado na Figura 4.4 como sendo aquele que está marcado a azul.

Para exemplificar melhor o preenchimento de uma *User Story Card,* vão ser transcritos as informações auxiliares geradas pela aplicação do 4SRS e que abarcam informações fundamentais para a geração da *User Story* correspondente. Como a documentação do projecto ISOFIN está escrita na língua inglesa, também os diagramas e as descrições o estão.

Caso de Uso. Começando a preencher o *User Story Card* a partir dos elementos mais "antigos" (aqueles que dão origem a outros), como acontece com o Caso de Uso, é necessário o próprio diagrama de Caso de Uso, assim como a descrição textual do mesmo. Como o 4SRS gera normalmente os AEs partindo dos Casos de Uso, é fácil obter-se essa documentação e extrair a informação relevante do Caso de Uso que gerou o AE, que por sua vez vai ser transformado numa *User Story*.

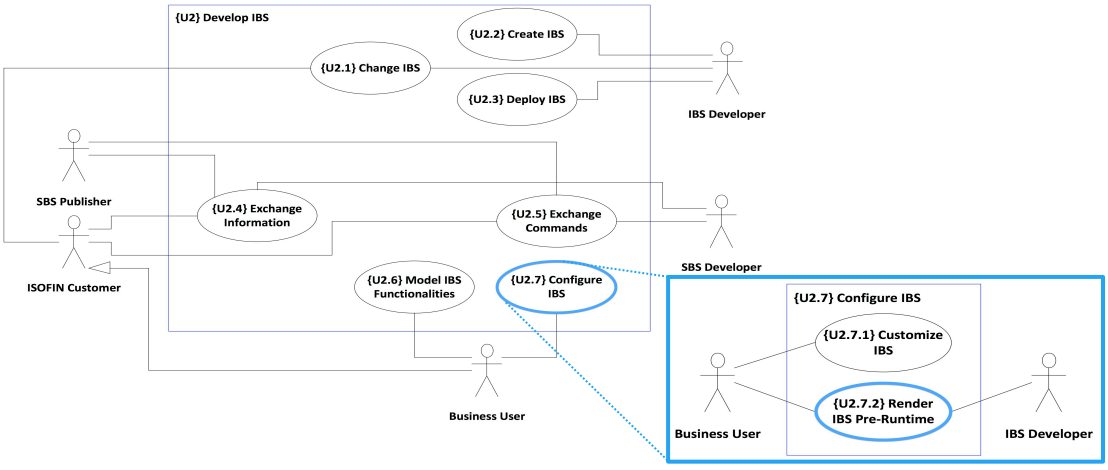

Figura 4.5 - Diagrama do Caso de Uso 2 e 2.7

A Figura 4.5 representa, a vários níveis de granularidade, os Casos de Uso que foram responsáveis pela geração do AE 2.7.2.c – o Caso de Uso 2.7.2 – *Render IBS Pre-Runtime*, que por sua vez descende do Caso de Uso 2.7 – *Configure IBS.* A Tabela 4.3, por sua vez, contém a descrição do Caso de Uso 2.7. Com os actores *Business User* e *IBS Developer* representados na figura acima e relacionados com o Caso de Uso em questão, juntamente com a descrição da tabela em baixo, já existe a informação necessária para preencher a *User Story Card*, ao que à informação relativa ao Caso de Uso diz respeito.

Use Case Name	Render IBS *Pre-Runtime*
Reference:	{U 2.7.2}
Description:	Configure and defines the pre-runtime of the IBS. This use case allows testing of the IBS before deployment. This use case will be required before the execution of {U2.3.2} Execute IBS Deployment to verify that no problems occur during the execution of the IBS.
Actors	Business User, IBS Developer
Special Requirements	Needs the existence of at least one SBS catalog
Pre-Conditions	Business User credential validation
Post-Conditions	{U 2.2.4} Create IBS Code

Tabela 4.3 - Descrição textual do Caso de Uso 2.7.2

AE. Relativamente ao AE, grande parte da informação que é necessária preencher para a *User Story Card,* encontra-se no próprio diagrama da arquitectura lógica da aplicação *IBS Management* (Figura 4.4 – AE representado a azul): o nome, o tipo e o módulo correspondente. A descrição do AE é gerada pela aplicação do 4SRS como pode ser visto na Tabela 4.4.

AE Name:	Test IBS Before Deployment
Reference:	{AE 2.7.2.c}
Description:	This AE allows testing of the IBS before deployment. This AE will be required before the execution of *{AE2.3.2.c} IBS Deployer* to verify that no problems occur during the execution of the IBS. All information need for the execution is provided by *{AE2.2.5.d} IBS Pre-Deployment Storage*

Tabela 4.4 - Descrição textual do AE 2.7.2.c

Relativamente às associações entre AEs, estas são identificáveis pela observação do diagrama lógico resultante do 4SRS. No entanto, como ambas as ligações (quer por associação directa, quer por ligação indirecta) são geradas no último passo do 4SRS, existe normalmente documentação auxiliar que faz referência a elas. Como a documentação gerada pela execução de cada passo é, em regra, não estruturada e contém informação não relevante para o processo FLAUS, optou-se por não ser incluída neste espaço.

user story	US #	*(gerado no passo 3)*		
	Critérios Aceitação	*		*Story Points*
				*
architectural element	AE #	2.7.2.c	Test IBS Before Deployment	
	Tipo	*Control*	Módulo	IBS Installer
	Descrição	This AE allows testing of the IBS before deployment. All information need for the execution is provided by {AE2.2.5.d} IBS Pre-Deployment Storage		
	Múltiplo	NÃO	Módulo	N/A
	Ligação	Assoc.	2.7.2.i – IBS Test Generator	
		Ind.	2.3.2.c – IBS Deployer 2.2.5.d – Pre-Deployment Storage	
caso de uso	CU #	2.7.2	Render IBS in Pre-Runtime	
	Descrição	Configure and defines the pre-runtime of the IBS. This use case allows testing of the IBS before deployment.		
	CU Asc.	2.7 – Configure IBS		
	Actores	Business User IBS Developer		
Esta *User Story Card* é apenas um exemplo e necessita da execução do 3º passo deste método para poder gerar o nome da *User Story*.				

Tabela 4.5 - *User Story Card* da *User Story* gerada a partir do AE 2.7.2.c

Mesmo sem passar para o último passo (geração da *User Story*), e observando cuidadosamente a Tabela 4.5, é possível constatar que a informação que vai permitir a

geração do nome da *User Story* está toda presente na tabela, quer seja relativa ao AE, quer seja relativa ao Caso de Uso correspondente.

3º Passo – Redigir User Story. Para redigir o nome de uma *User Story*, são necessários três informações distintas: qual é o actor beneficiado pela sua implementação ("quem" ou "*who"*), qual é a funcionalidade necessária (o "quê" ou "*what"*) e qual a necessidade da mesma ser implementada ("porquê" ou "*why"*).

Relativamente ao "quem", a *User Story Card* produzida no passo 2 tem a resposta, pela observação do actor que interage com o Caso de Uso que deu origem ao AE. Neste caso, o "quem" é um actor do tipo *Business User* ou *IBS Developer*.

Por ser uma *User Story* concebida a partir de um AE de controlo, é necessário extrair o tipo de acção que este representa, normalmente obtendo um verbo que caracterize a acção e informação adicional que permita perceber o alcance (ou escopo) da acção. Esta informação normalmente encontra-se no nome do AE, mas podem ser obtidos dados adicionais na descrição do AE, que pode completar (ou até mesmo substituir) alguns elementos da parte "o quê" da *User Story*. Neste exemplo, a escrita da parte central é relativamente simples: a acção que se quer implementar é um teste (que vem da palavra inglesa *test*) e o escopo é um objecto do tipo IBS, teste este que é necessário ocorrer na fase anterior ao *deployment* (*before deployment*). Juntando os actores envolvidos, já é possível perceber o que é que é necessário implementar e para quem: "*As a Business User or an IBS Developer, I want/need to test IBS before deployment*". As palavras de ligação costumam ser sempre as mesmas, e referem-se ao facto de um determinado actor necessitar ou querer (em inglês *"I want"* ou "*I need"*) executar uma determinada acção. Existem, no entanto, outras expressões que ligam as três secções de uma *User Story*, como por exemplo "necessito de ter" (*"need to have"),* que permita ("*be able to"*), entre outras.

A última secção da *User Story* refere-se ao motivo pelo qual uma determinada funcionalidade é necessária no produto a implementar. Para este detalhe, é necessário voltar ao componente que deu origem a todos estes diagramas e elementos e que corresponde às verdadeiras necessidades do utilizador: o Caso de Uso. Embora em alguns nomes de AE se perceba a motivação para os mesmos existirem, os Casos de Uso normalmente possuem informação extra, que está sempre ligada ao actor que também está presente no início da *User Story*. Neste caso de exemplo, o nome do Caso

de Uso ("*Render IBS in Pre-Runtime*") é o verdadeiro motivo pelo qual é necessária executar a acção de teste detalhada na *User Story*: para renderizar um IBS na fase *pre-runtime*. Na eventualidade do nome do Caso de Uso não ser esclarecedor ou específico o suficiente, é possível procurar na descrição do AE uma possível motivação para a implementação daquela funcionalidade. A descrição refere que esta funcionalidade de teste é importante de modo a verificar, antes da entrega do IBS, se ocorrem problemas durante a execução do IBS (tradução retirada do seguinte conteúdo: *"This AE will be required before the execution of {AE2.3.2.c} IBS Deployer to verify that no problems occur during the execution of the IBS.").*

Assim, e usando a língua original de toda a documentação, já se gerou o nome da *User Story,* a peça que faltava para completar o *User Story Card* construído no passo 2.

> "As a Business User or an IBS Developer, I want/need to test an IBS before deployment, in order to render IBS in pre-runtime."

Como pode ser observado no nome da *User Story*, por vezes é necessário fazer pequenas alterações no texto de modo a formar uma frase que faça sentido sintática e semanticamente na língua em que é escrita. Pontuação (acréscimo de vírgulas, por exemplo) ou palavras de ligação (como complementos directos ou indirectos) são exemplos recorrentes de palavras acrescentadas para darem sentido ao nome.

AEs de *interface* e de *data*. Para exemplificar totalmente a execução do terceiro passo do processo FLAUS, falta a sua aplicação sobre AEs que sejam do tipo *interface* ou do tipo de *data*. É no terceiro passo que existem algumas diferenças provenientes do tipo de AE, que produz lógicas um pouco díspares nos AE de interface e *data* dado que, pelo próprio tipo de AE, já é possível descortinar o tipo de funcionalidade necessária.

Para exemplificar estes dois tipos diferentes, foram escolhidos os seguintes AEs: 2.1.3.d – *IBS Structure Repository* e o 2.1.3.i – *IBS Structure Interface*, respectivamente do tipo *interface* e do tipo *data*. Nestes casos, foi produzido uma *User Story Card* contendo apenas a informação necessária para a execução do terceiro passo, descartando a informação que serve apenas para auxiliar a equipa na estimativa e implementação.

Tal como já foi referido no capítulo anterior, os AEs do tipo *interface* e *data* têm ambos uma característica especial que advém do próprio tipo de AE e que simplifica o processo da escrita da *User Story.* É possível perceber, só por olhar para o identificador do AE, que estas *User Stories* se referem a uma conexão a uma fonte informação (AE 2.1.3.d - Tabela 4.7) ou que se referem à necessidade de existir uma *interface* que permita a troca de informação (AE 2.1.3.i - Tabela 4.6).

us	US #	*(gerado no passo 3)*		
architectural element	AE #	2.1.3.i	Update IBS Structure Interface	
	Tipo	*Interface*	Módulo	IBS Editor
	Descrição	Shows the IBS structure, created in {AE2.2.3.i} IBS Structure interface and retrieved on {AE2.1.1.d} IBS Repository, and allows defining of the new IBS structure by dragging and dropping existing IBS or SBS icons from the palette into the design area. Also allows creating connections between them. These connections may result in transformations of the interfaces in order to permit the interoperability. The transformations are manually created. Provides a pallet with all functionalities of the SBS and IBS. The objects provided from the pallet are automatically created with the information provided by {AE2.6.1.i1} IBS Pallet Retrieval Interface and {AE2.6.2.i1} SBS Pallet Retrieval Interface.		
caso de uso	CU #	2.1.3	Change IBS Structure	
	Descrição	The IBS model shows which SBS/IBS makes it and how they are connected. The IBS structure to be change was created in {U2.2.3} Model IBS Structure, this model is retrieved from the execution of {U2.1.1} Load IBS. Shows the IBS structure defined in {U2.2.3} Model IBS Structure and allows defining of the new IBS structure by dragging and dropping existing IBS or SBS icons from the palette into the design area. Also allows creating connections between them. These connections may result in transformations of the interfaces in order to permit the interoperability. The transformations are manually created.		
	CU Asc.	2.1 – Change IBS		
	Actores	ISOFIN Customer / IBS Developer		

Tabela 4.6 - User Story Card da User Story gerada a partir do AE 2.1.3.i

us	US #	*(gerado no passo 3)*		
architectural element	AE #	2.1.3.d	IBS Structure Repository	
	Tipo	*data*	Módulo	IBS Editor
	Descrição	Contains all information from the model created in {AE2.1.3.i} Update IBS Structure Interface. This information will be need during the execution of {AE2.1.4.c} Compiles IBS information.		
caso de uso	CU #	2.1.3	Change IBS Structure	
	Descrição	The IBS model shows which SBS/IBS makes it and how they are connected. The IBS structure to be change was created in {U2.2.3} Model IBS Structure, this model is retrieved from the execution of {U2.1.1} Load IBS. Shows the IBS structure defined in {U2.2.3} Model IBS Structure and allows defining of the new IBS structure by dragging and dropping existing IBS or SBS icons from the palette into the design area. Also allows creating connections between them. These connections may result in transformations of the interfaces in order to permit the interoperability. The transformations are manually created.		
	CU Asc.	2.1 – Change IBS		
	Actores	ISOFIN Customer / IBS Developer		

Tabela 4.7 - User Story Card da User Story gerada a partir do AE 2.1.3.d

Assim, e começando pelo AE de *data*, com o título de "*IBS Structure Repository*", é relativamente simples compreender que este AE refere a necessidade de existir um repositório de armazenamento de um certo tipo de informação, neste caso de estruturas de IBS. Continuando a utilizar a forma de ligação "*I want / I need*", é necessário exprimir a necessidade da existência de um repositório, através da forma verbal eu necessito/quero ter, seguido do repositório em questão. Relativamente à parte da justificação da necessidade (a parte *why* da *User Story*), o nome do Caso de Uso de onde descende o AE é completamente auto-explicativo, referindo-se à necessidade de poder mudar a estrutura dos IBS.

Juntando todos os pedaços da *User Story*, obtém-se a seguinte designação:

> "As an ISOFIN Customer or an IBS Developer, I want/need to have an IBS structure repository, in order to change IBS structure."

Para a *User Story* produzida pela aplicação do processo FLAUS no AE 2.1.3.i, a lógica é idêntica ao caso anterior. Ao invés da necessidade de ter uma camada de

armazenamento e/ou acesso aos dados, é necessária uma *interface* para trocar informação ou disponibilizar conteúdos. Partindo disso, é possível afirmar que os utilizadores alvo (*ISOFIN Customer* ou *IBS Developer*), necessitam de uma *interface* para comunicar, neste caso para actualizar a informação relativa às estruturas de IBS alteradas. Como este AE também foi gerado pela aplicação do 4SRS no Caso de Uso *Change IBS*, também a última parte da *User Story* é idêntica ao caso anterior. Formula-se então a seguinte denominação:

> "As an ISOFIN Customer or an IBS Developer, I want/need to have an IBS structure repository, in order to change IBS structure."

Utilizando o processo FLAUS, é possível produzir uma lista de *User Stories* geradas a partir dos AEs da aplicação *IBS Management* e que aqui estão representadas em forma de tabelas. A *User Story* é formada pela conjunção do cabeçalho da terceira coluna com a informação da terceira célula da linha correspondente para a parte "quem", seguido do cabeçalho da 4ª coluna e da célula correspondente para a parque "quê", finalizado com a informação do cabeçalho da 5ª coluna com a célula correspondente para a parte "porquê". Adicionalmente, podem-se utilizar expressões de ligação para conectar as diferentes secções, como exemplificado anteriormente.

Apesar de todos os AEs do módulo escolhido terem gerado as correspondentes *User Stories* através da aplicação do processo FLAUS, detalhado no capítulo anterior, existiram casos onde foram necessárias algumas adaptações manuais de modo a conferir significado sintático e semântico às *User Stories* geradas. As *User Stories* das restantes aplicações do ISOFIN podem ser consultadas nos Anexos II a VII deste trabalho.

AE #	Nome	As a(n) <actor>	I want/need <description>	In order to <outcome>
2.1.2.c1	Selected Object Configurations	ISOFIN Customer / IBS Developer	selected object configurations	change (BS Structure configurations
2.1.4.c	Compiles IBS information	IBS Developer	compile IBS changes and information	create a new IBS
2.2.4.c	Define IBS Code Gaps	IBS Developer	automatically define IBS code gaps	create IBS code
2.2.5.c	Compile IBS code	IBS Developer	compile IBS code	create a new IBS
2.2.7.c	IBS Interface Generator	IBS Developer	IBS interface generator	generate interface for the IBS usage based on the generated code
2.3.2.c	IBS Deployer	IBS Developer	IBS deployer	execute IBS deployment
2.7.1.c	IBS Customization Filter	Business User	IBS customization filter	customize and configure IBS
2.7.2.c	Test IBS Before deployment	Business User / IBS Developer	test IBS before deployment	render IBS Pre-Runtime

Tabela 4.8 – Lista de User Stories geradas a partir de AE do tipo *control*

AE #	Nome	As a(n) <actor>	I want/need <description>	In order to <outcome>
2.1.1.d	IBS Repository	ISOFIN Customer / IBS Developer	IBS repository	store all meta-information related with IBS and Load IBS from the catalog
2.1.2.d	IBS Configuration Repository	ISOFIN Customer / IBS Developer	IBS configuration repository	change IBS configurations
2.1.3.d	IBS Structure Repository	ISOFIN Customer / IBS Developer	IBS structure repository	change IBS structures
2.2.5.d	IBS Pre-Deployment Storage	IBS Developer	IBS pre-deployment storage	store IBS catalog compiled IBS Code
2.2.7.d	IBS Interface Repository	IBS Developer	IBS interface repository	store the generated IBS interfaces
2.6.2.d	SBS Repository	Business User	SBS repository	store and retrieve SBS information

Tabela 4.9 – Lista de User Stories geradas a partir de AE do tipo *data*

AE #	Nome	As a(n) <actor>	I want/need <description>	In order to <outcome>
2.1.2.i	Low-level IBS Configuration Interface	ISOFIN Customer / IBS Developer	low-level IBS configuration interface	provide interface to change IBS configurations
2.1.3.i	IBS Structure Interface	ISOFIN Customer / IBS Developer	IBS structure Interface	show the IBS Structures
2.1.4.i	Update IBS Interface	IBS Developer	update IBS interface	verify IBS's listo f changes and create a new IBS
2.2.3.I	IBS Structure Interface	IBS Developer	IBS Structure Interface	model IBS structure
2.2.4.i	IBS IDE	IBS Developer	IBS IDE interface	create IBS code
2.2.6.i	Permissions Interface	IBS Developer	Permissions Interface	set and manage permissions
2.3.1.i	IBS Publisher Interface	IBS Developer	IBS Publisher Interface	publish IBS in catalog and deploy IBS
2.3.2.i	IBS Deployer Interface	IBS Developer	IBS Deployer Interface	execute IBS deployment
2.3.3.i	IBS Documentation Editor	IBS Developer / IBS Business Analyst	IBS Documentation Editor Interface	allow verification of the generated IBS documentation

Tabela 4.10 – Lista de User Stories geradas a partir de AE do tipo *interface*

4.3 Adaptações manuais às *User Stories* do Projecto ISOFIN

Embora na maior parte dos casos os nomes dos conjuntos de AE/Caso de Uso relacionados tenham informação suficiente para compor uma denominação clara da *User Story*, existem casos onde é necessário modificar gramaticalmente a frase para fazer sentido, acrescentar informação à parte intermédia (parte "*what"*) ou parte final (parte "*why"*) para que o nome seja entendível sem necessitar de aceder a informações auxiliares.

Seguidamente, vão ser listadas algumas modificações necessárias para gerar as *User Stories* das Tabelas 4.8, 4.9 e 4.10.

Nome de AE idêntico ao nome do Caso de Uso que lhe deu origem. Existem casos onde o processo de 4SRS não gera AEs de nome diferente ao Caso de Uso que lhes dá origem, como acontece, por exemplo, com o AE 2.2.5.c – *Compile IBS*

Code, descendente do Caso de Uso 2.2.5 – *Compile IBS Code*. Neste caso, se fosse aplicado o processo FLAUS sem qualquer adaptação, a *User Story* que surgiria seria: "*As an IBS Developer, I want to compile IBS code in order to compile IBS code*". Para evitar a redundância nestes casos, uma das técnicas a empregar é a utilização do Caso de Uso mais alto nível, de onde descende o Caso de Uso que gerou um AE pela aplicação do 4SRS. O Caso de Uso 2.2.5 é uma desagregação do Caso de Uso 2.2 – *Create IBS*, como pode ser visto no campo da *User Story Card* referente ao AE 2.2.5.c. Assim, é possível manter a denominação do AE na secção central da *User Story*, e deixar para a parte justificativa a informação herdada do Caso de Uso ascendente. Desta forma, utilizando a informação do Caso de Uso 2.2, o nome da *User Story* será:

> "As an IBS Developer, I want/need to compile IBS code, in order to create a new IBS."

Embora a *User Story* pareça ser um pouco genérica devido ao facto de utilizarmos informação de Casos de Uso mais alto nível, a informação presente na respectiva *User Story Card* irá ajudar a perceber toda a extensão da funcionalidade pretendida.

Acréscimo de informação à parte *why* da *User Story*. Seguindo a lógica do exemplo anterior, também na parte justificativa da necessidade da implementação da funcionalidade para um determinado utilizador, por vezes é positivo acrescentar uma ou duas palavras que ajudem na explicação mencionada. Este caso pode ocorrer mais frequentemente, tudo dependendo do critério do analista que está a aplicar o processo, caso entenda que é oportuno acrescentar um pouco mais de informação à denominação da *User Story*. Esta informação extra pode ser retirada da descrição do AE ou do Caso de Uso e ainda se pode acrescentar informação do Caso de Uso de alto nível, como já se verificou anteriormente.

Exemplos disso podem ser encontrados nas *User Stories* dos AEs 2.1.2.d – *IBS Configuration Repository* ou 3.1.i – *ISOFIN Model Editor Interface*. No primeiro caso – "*As an ISOFIN Customer / IBS Developer have an IBS Configuration Repository in order to change (IBS) configurations".* Neste exemplo, é oportuno acrescentar o termo IBS para ser reforçada a ideia de que é possível alterar as configurações de um determinado IBS.

Também no segundo exemplo – "*As an IBS Business Analyst, I want/need to have an ISOFIN App Model Editor, in order to model (the composition of) an ISOFIN*

application", é oportuno especificar que esta funcionalidade se refere à modelação da composição interna de uma aplicação ISOFIN.

A aplicação destas alterações, embora não sejam obrigatórias para manter a coerência exigida para um nome de uma *User Story,* conferem uma explicação extra que pode ser importante quando se está a listar *User Stories* num *Backlog,* ou quando não se tem acesso à respectiva *User Story Card*.

Alterações às formas verbais. Existem ainda situações onde o nome do 4SRS, juntamente com a estrutura base do nome de uma *User Story* ("Como um" / "*As a*"; "de modo a" / "*in order to*"), não produzem *User Stories* gramaticalmente coerentes. Nestes casos, é por vezes necessário proceder a alterações das formas verbais ou a pequenas alterações na ordem das palavras, de modo a conseguir um nome que faça sentido gramaticalmente e ainda mantenha o seu significado técnico.

Um caso exemplificativo é a *User Story* do AE 2.1.4.c – *Compiles IBS information*. Neste caso, deve-se mudar a forma do verbo *To Compile* da terceira pessoa do presente do indicativo (*compiles*) para a primeira (*compile*), para encaixar na estrutura base de uma *User Story*, onde é tudo relacionado com o actor beneficiado pela *User Story*.

Embora estas alterações possam ser consideradas menores, são importantes para acrescentar coerência às *User Stories* geradas. Fica à responsabilidade final do analista que está a especificar os requisitos em forma de *User Stories,* a correcção de eventuais nomes que possam incluir algumas redundâncias ou falhas gramaticais.

4.4 O *Scrum* e o ISOFIN

Apesar das *User Stories* já estarem geradas e terem associadas informação necessária para proceder a uma estimativa cuidada, existem outros pontos que são necessários afinar, de modo a que várias equipas de implementação possam trabalhar em simultâneo neste projecto, utilizando uma metodologia ágil. Embora existam alguns casos de estudo relativos a equipas de *Scrum* distribuído a trabalhar paralelamente no mesmo produto, a coordenação entre equipas continua um dos tópicos que mais discussão gera e onde os avanços e progressos são mais difíceis e lentos de atingir [Begel et al., 2009].

Apesar das dificuldades inerentes a essa coordenação, através do conjunto definido de práticas, artefactos e eventos que caracterizam o *Scrum* (secção 2.2), esta não deixa de ser uma metodologia ágil que tem a flexibilidade suficiente de se poder adaptar ao tipo de equipas/projectos em questão, de modo a conseguir potenciar ao máximo o rendimento dos seus elementos. Estas adaptações são necessárias devido à particularidade de se querer implementar simultaneamente as aplicações (reduzindo assim o *Time to Market*) e ainda devido ao facto das *User Stories* e arquitectura do sistema terem sido geradas através da aplicação do 4SRS, combinada com o processo FLAUS.

Equipa e Papéis. Embora não seja conhecido (nem relevante) o exacto número de equipas de implementação, existem sugestões que podem ser consideradas, partindo do princípio que os elementos da equipa pertencem à mesma organização. A sua localização também é irrelevante, partindo-se do princípio de que os elementos da equipa podem trabalhar remotamente, independentemente do local ou fuso horário a que pertencem.

O *Scrum Master*, elemento que tem como principal responsabilidade a manutenção da produtividade da equipa, pode ser simultâneo às várias equipas, permitindo uma percepção do estado de todas as aplicações a serem implementadas. Além de isso significar uma maior concentração de recursos humanos focados na implementação, este elemento, sempre em conjunto com o *Product Owner* (que neste caso vai acumular o papel com o de Gestor Técnico – obrigatório neste tipo de projectos), pode sugerir a realocação de elementos de desenvolvimento, de maneira a conseguir manter as equipas alinhadas e paralelas no seu esforço de implementação.

Como o projecto ISOFIN é um projecto de fundos europeus, por vezes o contacto com o preponente do projecto pode ser complicado e demorado, pelo que é necessário encontrar um elemento que, embora seja a *interface* de comunicação com o preponente, tem um papel mais activo na prioritização do trabalho e nas decisões mais técnicas da solução. Esse elemento, além de poder realocar recursos entre as equipas, necessita de acompanhar de perto o trabalho de implementação, de modo a conseguir rapidamente detectar potenciais atrasos ou problemas críticos no processo de implementação do produto. Neste caso, o gestor técnico da solução pode ser

considerado como *Product Owner*, dado que é a ele que cabem as principais decisões no decorrer do processo de implementação.

***Sprints* e Eventos.** Uma das principais características deste projecto é a implementação em simultâneo de várias equipas de desenvolvimento da mesma solução, com recursos partilhados (caso do *Scrum Master*, por exemplo), o que vai obrigar a uma grande flexibilidade por parte das equipas.

A primeira sugestão refere-se ao facto das *Sprints* de todas as equipas, embora com a mesma duração, estejam dessincronizadas relativamente à data de início, de maneira que o *Scrum Master* e o *Product Owner* tenham total disponibilidade para trabalhar com todas as equipas equitativamente. Além disso, caso existam problemas reportados por uma equipa na *Sprint Review* ou *Sprint Retrospective*, existe tempo suficiente para re-calendarizar aspectos nas outras equipas ou ainda realocar recursos no final dos sprints das outras equipas.

´De maneira a que todas as equipas estejam por dentro do que se vai passando em todas as aplicações do projecto, é considerada ainda a possibilidade de um elemento de cada equipa esteja presente na *Sprint Review* de outra equipa. Esse elemento externo deve ser sempre diferente, de modo a que todas as pessoas da equipa de implementação tenham oportunidade de conhecer o trabalho realizado nas diversas equipas.

***Grooming*.** É um dos eventos do *Scrum* com mais preponderância neste projecto, por diversas razões. Por um lado, a geração automática de *User Stories* faz com que as mesmas tenham de ser revistas por toda a equipa, estimadas na sua complexidade de implementação (em *Story Points*) e apresentadas a toda a equipa para que todos conheçam claramente o que é necessário implementar.

Por outro lado, o processo FLAUS e o 4SRS omitem os requisitos não funcionais do projecto. Mesmo o *Scrum* é uma abordagem metodológica que por vezes tende a deixar estes detalhes para segundo plano, e é responsabilidade de toda a equipa criar tarefas que salvaguardem estas questões, ou que incluam dentro das *User Stories* que se podem relacionar com esses requisitos não funcionais.

Finalmente, o 4SRS gera arquitecturas onde a comunicação entre os AEs é deveras importante e fundamental para cumprir as necessidades do cliente, além de que existem AEs (e consequentemente *User Stories)* que são partilhadas por mais do que uma aplicação. É função do gestor técnico, juntamente com a equipa responsável pela implementação dessa *User Story* partilhada (equipa responsável pelo módulo a que pertence o AE originalmente), a comunicação e geração de documentação necessária para que as outras equipas (que também dependem dessa *User Story)*, possam continuar a implementar em paralelo. Aqui, a geração de critérios de aceitação que garantam o princípio *Independent* da *User Story* é muito importante, especificação essa que deve ser feita e apresentada à equipa no espaço do *Grooming*.

4.5 Conclusão

Apesar de existir grande espaço para melhorias no processo FLAUS, fica exemplificado que o mesmo pode ser adaptado em projectos de grande envergadura e complexidade. Os passos claramente definidos, com *templates* preparados para serem preenchidos, torna a sua aplicação relativamente fácil por alguém que tenha experiência na área da Engenharia do *Software*, nomeadamente na gestão de requisitos.

No entanto, embora as *User Stories* estejam delineadas e a informação a si acoplada, este processo ainda carece de uma revisão e especificação mais contextualizada, por parte da equipa de desenvolvimento. Estimativas temporais e de complexidade, eventuais desfragmentações de *User Stories* que porventura não sigam à risca os princípios INVEST (nomeadamente na parte do *small* – curta - e do *estimable* - estimável) e a definição dos critérios de aceitação a partir da informação presente no *User Story Card* são ainda passos necessários antes da *User Story* entrar no processo normal de *Sprint*.

Paralelamente, a gestão de projecto não se esgota na especificação detalhada dos requisitos a implementar. A aplicação do próprio *Scrum* em diversas equipas que estão a implementar aplicações para o mesmo produto precisa de ser bem afinado, de maneira a conseguir potenciar a produtividade das equipas enquanto se vão mitigando eventuais pontos de falha e de trabalho sobreposto das equipas. As recomendações feitas para as equipas *Scrum* do projecto ISOFIN foram adaptadas dos principais referenciais do *Scrum*, embora adaptados para o projecto em questão. A existência de muitos AEs que pertencem a diversas aplicações e a falta de um *Product Owner* definido serão os

maiores desafios que as equipas terão de ultrapassar. Nestes casos, os momentos de *Sprint Retrospective,* onde a equipa pode melhorar e adaptar a própria aplicação do *Scrum* vão ser fundamentais, dado que serão necessários vários ajustamentos até se atingir a formula óptima que funcione para implementar o projecto ISOFIN no mais curto espaço de tempo possível. O ideal "inspecionar e adaptar" aparenta ser de uma importância extrema neste contexto.

5. Conclusões

A temática da gestão de projectos de *software*, com todos os contextos associados, não é uma questão binária de métodos ágeis contra métodos tradicionais. Embora exista uma clara disrupção de pensamentos entre ambas as tendências, existem pontos de contacto e de correlação que devem ser explorados e adaptados de modo a conjugar o melhor dos dois mundos. A necessidade de especificação formal e de definição de arquitecturas sólidas na fase de *Design* dos métodos mais tradicionais pode ser combinada com a iteratividade e menor formalismo dos métodos ágeis de *software*. O processo FLAUS tenta introduzir a variável referente à agilidade num mecanismo (4SRS) mais vocacionado à sua integração nos métodos mais tradicionais e formais.

Por sua vez, o *Scrum* é uma metodologia tão leve e desprovida de referências relacionadas com a construção do produto que nem sempre é considerada uma metodologia. Frequentemente, o *Scrum* é apelidado de *framework* de princípios e práticas que definem a maneira como as equipas devem trabalhar e interagir entre si [Keith, 2008]. No seu referencial, o *Scrum* indica eventos, papéis e artefactos que, se utilizados de forma correcta, conseguem praticamente auto-gerir uma equipa que tenha um elevado grau de comprometimento relacionado com o trabalho a executar.

Relativamente aos objectivos propostos neste trabalho, o *Scrum* foi analisado com extensivo detalhe, oferecendo uma panorâmica de funcionamento de um ciclo iterativo, ou *Sprint*, explicando as responsabilidades de cada elemento na equipa, detalhando os artefactos que a mesma pode utilizar para tornar a sua *Sprint* mais produtiva, e ainda explicando a cadência e estrutura de cada evento necessário para completar o ciclo. Foi dada especial atenção à gestão de requisitos em equipas que seguem a abordagem do

Scrum, com uma apresentação do formato das *User Stories*, nomeadamente as suas características, a sua sintaxe e as informações que se podem associar a estes requisitos.

Adicionalmente, foi especificado um modelo processual que permite a transformação de arquitecturas lógicas em requisitos compatíveis com as equipas de *Scrum*, no formato de *User Stories*. Foi produzido o processo FLAUS, que é executado sobre o resultado final da aplicação do 4SRR, que consiste em três passos distintos. Inicialmente, são geradas tabelas agregadoras de AEs por tipo e aplicação. De seguida, são preenchidas *User Story Cards* que contêm a informação relativa à *User Story* gerada no último passo, informação essa que provém da aplicação do 4SRS e que pode ajudar as equipas a implementar as referidas funcionalidades.

Por fim, foi testada a aplicação do modelo criado num projecto real. O ISOFIN, projecto de fundos comunitários, no qual a Universidade do Minho está inserida com as suas valências na fase de especificação de requisitos e geração de arquitecturas do produto, foi o escolhido no âmbito deste trabalho. Aqui se provou que as *User Stories* podem ser geradas como preconizado no capítulo 3, com a informação relativa a cada uma armazenada na respectiva *User Story Card*. Paralelamente, foi sugerida uma adaptação do *Scrum* ao caso específico do ISOFIN, onde múltiplas equipas vão trabalhar em simultâneo, e onde a sua comunicação e sincronização é fundamental. Neste caso, a proposta apresentada carece de validação no terreno, de modo a perceber se o modelo sugerido de equipas a trabalhar em simultâneo partilhando o *Scrum Master,* conjugado com a existência do papel de *Product Owner* acumulado com o de Gestor Técnico, põem em causa a eficiência desta abordagem metodológica. Recorde-se que o *Scrum* já foi provado como eficiente em diversos casos de estudo, desde equipas e empresas pequenas, até grandes empresas ou produtos com diversas equipas a implementar em simultâneo.

A conclusão final a que se pode chegar é que é possível a inserção de arquitecturas lógicas em equipas puramente formatadas para usar o *Scrum*. Existe espaço para a aplicação do processo FLAUS que permite a geração de *User Stories,* utilizando toda a informação útil recolhida por processos anteriormente aplicados e já validados por múltiplas utilizações no contexto profissional.

5.1 Limitações do processo FLAUS e sua aplicação no Projecto ISOFIN

Apesar dos objectivos a que este trabalho se propôs terem sido respondidos, essas mesmas respostas não surgiram desprovidas de limitações que necessitem de trabalho suplementar para conferir robustez ao processo criado.

Adaptações semânticas ou de conteúdo. A principal limitação relativa ao processo FLAUS em si é a eventual necessidade de corrigir semanticamente o nome da *User Story* gerada no terceiro passo. Seja devido ao nome do AE que lhe deu origem ser ambíguo, seja pelo nome do Caso de Uso relacionado ser semelhante ao do AE gerado, existem diversos casos onde o analista responsável pela aplicação do processo tem de proceder a alterações manuais. Esta limitação, embora facilmente ultrapassável pelo recurso às informações armazenadas na *User Story Card,* é a maior falha relacionada com o processo proposto.

Validação no terreno. Embora exista a aplicação do processo FLAUS no contexto real, a sua aplicação não passou ainda do papel, carenciando de uma validação no terreno, com monitorização constante sobre a adequabilidade das *User Stories* ao trabalho necessário e à equipa de desenvolvimento. Esta falta de validação pode originar alterações no formato e na estrutura das *User Story Cards*, caso se asserte que existe informação prescindível, ou que, pelo contrário, falte informação que facilite o processo de estimação e planeamento da implementação da *User Story.*

Inclusivamente, a aplicação do processo FLAUS por um analista de sistemas experiente, com conhecimento relativamente à gestão de requisitos em projectos que seguem abordagens ágeis, poderia significar um conjunto de importante contributos que, além de validar o resultado final do processo, poderia resultar em várias sugestões que simplificassem e concedessem outras valências ao processo em questão.

Dependência do 4SRS. O processo FLAUS necessita, para a sua aplicação rápida e competente, de informação que é gerada apenas pela aplicação do processo de 4SRS, tal como indicado pelos seus autores. Este facto deixa a utilização deste processo dependente não só da informação do 4SRS, como também da existência de condições

favoráveis para a aplicação do próprio 4SRS. A utilização do processo FLAUS apenas sobre o diagrama de componentes (sem recorrer à informação adicional) é possível, mas as *User Story Cards* geradas não possuiriam muita da informação considerada essencial para a estimativa e implementação por parte da equipa de desenvolvimento. Por outro lado, eventuais *User Stories* com pouco sentido semântico não poderiam recorrer a essa mesma informação auxiliar, de modo a corrigir e acrescentar informação que lhe conferisse mais sentido.

5.2 Trabalho futuro

De modo a contrariar as limitações indicadas existe um conjunto de tarefas futuras que, quando executadas, poderão conferir uma maior maturidade ao processo FLAUS.

Automatização. Dado que a aplicação do processo FLAUS depende de informação que pode ser estruturada e de diagramas lógicos que podem ser decompostos através de ferramentas destinadas para o efeito, abre-se espaço para a criação de um mecanismo que automatize o processo FLAUS. Isso pouparia bastante esforço de consulta de documentação e agregação da informação na *User Story Card*. Apesar disso, o facto de existirem muitos casos de geração de *User Stories* com problemas semânticos quando o processo FLAUS é aplicado, tornaria obrigatória uma revisão do resultado final da aplicação do mesmo.

Aplicação em múltiplos projectos. Como foi referido nas limitações deste trabalho, para que o processo FLAUS seja validado, é necessário a sua aplicação no âmbito organizacional, com monitorização dos resultados e análise das percepções dos *Scrum Masters* e *Product Owner* das equipas envolvidas. Apenas com a utilização sucessiva deste processo e em âmbitos tecnológicos diferentes é que pode afiançar, com certeza, que o mesmo se apresenta como uma alternativa válida para a especificação de *User Stories* para consumo de equipas *Scrum*.

Geração de User Stories a partir de Casos de Uso. De modo a mitigar a necessidade da aplicação correcta do 4SRS para ser possível o uso do processo FLAUS, abre-se espaço a uma pesquisa que explore a possibilidade de se adaptar o

processo FLAUS a um contexto mais simples, apenas composto inicialmente por um conjunto de Casos de Uso e de actores. Na grande maioria dos casos, nomeadamente nos projectos mais curtos e com menor grau de complexidade, os Casos de Uso são a única representação gráfica dos requisitos funcionais do produto, seguindo-se logo a sua implementação. A possibilidade de ser aplicado o processo FLAUS, mesmo que isso implicasse a não geração de uma arquitectura lógica e a acoplação de menos informação à *User Story Card*, abriria o leque de projectos onde este poderia ser aplicado.

6. Referências Bibliográficas e Bibliografia

Ambler, S., & Lines, M. (2012). *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*. IBM Press.

Azevedo, S., Machado, R., Muthig, D., & Ribeiro, H. (2009). Refinement of Software Product Line Architectures through Recursive Modeling Techniques. In *On the Move to Meaningful Internet Systems: OTM 2009 Workshops* (pp. 411–422). Springer.

Baskerville, R., Ramesh, B., Levine, L., Pries-Heje, J., & Slaughter, S. (2003). Is Internet-Speed Software Development Different? *Software, IEEE*, *20*(6), 70–77.

Bates, C., & Yates, S. (2008). Scrum Down: A Software Engineer and a Sociologist Explore the Implementation of an Agile Method. In *Proceedings of the 2008 International Workshop on Cooperative and Human Aspects of Software Engineering* (pp. 13–16). Leipzig, Deutschland: ACM Press.

Beavers, P. (2007). Managing a Large "Agile" Software Engineering Organization. In *Procedings of the Agile Conference - AGILE '07* (pp. 296–303). Washington DC, USA: IEEE.

Begel, A., & Nagappan, N. (2007). Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study. In *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement - ESEM '07* (pp. 255–264). Madrid, España: IEEE.

Begel, A., Nagappan, N., Layman, L., & Poile, C. (2009). Coordination in the Large-Scale Software Teams. In *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering - CHASE '09* (pp. 1–7). Vancouver, Canada: IEEE.

Berczuk, S. (2007). Back to Basics: The Role of Agile Principles in Success with an Distributed Scrum Team. In *Procedings of the Agile Conference - AGILE '07* (pp. 382–388). Washington DC, USA: IEEE.

Boehm, B. (2000). Project Termination Doesn't Equal Project Failure. *Computer*, *33*(9), 94–96.

Boehm, B. (2002). Software Engineering Is a Value-Based Contact Sport. *IEEE Software*, *19*(5), 95–96.

Boehm, B. (2007). A Survey of Agile Development Methodologies. Retrieved October 10, 2012, from http://www.cems.uwe.ac.uk/~pchatter/2011/isd_hk/AgileMethods.pdf

Bosch, J., & Molin, P. (1999). Software architecture design: evaluation and transformation. In IEEE (Ed.), *Proceedings of the 1999 IEEE Conference and Workshop on Engineering of Computer-Based Systems - ECBS'99* (pp. 4–10). Nashville, USA: IEEE Comput. Soc.

Brown, N., Nord, R., & Ozkaya, I. (2010). Enabling Agility Through Architecture. *Crosstalk*, 12–17.

Cardozo, E., Neto, J., Barza, A., França, A., & Silva, F. (2010). Scrum and Productivity in Software Projects: A Systematic Literature Review. In *14th International Conference on Evaluation and Assessment in Software Engineering - EASE* (pp. 1–4). Keele, United Kingdom: British Computer Society.

Castro, J., Kolp, M., & Mylopoulos, J. (2002). Towards Requirements-Driven Information Systems Engineering: The Tropos Project. *Journal of Information Systems*, *27*(6), 365–389.

Cervone, H. (2011). Understanding agile project management methods using Scrum. *OCLC Systems & Services*, *27*(1), 18–22.

Cho, J. (2008). Issues and Challenges of Agile Software Development with Scrum. *Issues in Information Systems*, *9*(2), 188–195.

Cohn, M. (2003). Toward a Catalog of Scrum Smells. *Scrumalliance.org*. Retrieved October 17, 2012, from http://www.scrumalliance.org/resources/22

Collaris, R., Dekker, E., & Veen, J. Van. (2010). Scrum in a Traditional Project Organization. *Agile Record*, 8–12.

Cristal, M., Wildt, D., & Prikladnicki, R. (2008). Usage of Scrum Practices within a Global Company. In *IEEE International Conference on Global Software Engineering - ICGSE 2008* (pp. 222–226). Bangalore, India: IEEE.

Deemer, P., Benefield, G., Larman, C., & Vodde, B. (2010). *The Scrum Primer* (pp. 1–22). Retrieved from http://www.brianidavidson.com/agile/docs/scrumprimer121.pdf

Dingsøyr, T., Hanssen, G., & Dybå, T. (2006). Developing software with scrum in a small cross-organizational project. In *Proceedings of Software Process Improvement - EuroSPI '06* (pp. 5–15). Joensuu, Finland: Springer. Retrieved from http://link.springer.com/chapter/10.1007/11908562_2

Dominguez, J. (2009). *The Curious Case of the CHAOS Report 2009*. *Project Smart*. Retrieved from http://www.projectsmart.co.uk/pdf/the-curious-case-of-the-chaos-report-2009.pdf

Fernandes, J., & Machado, R. (2001). From use cases to objects: an industrial information systems case study analysis. In *Proceedings of the Seventh International Conference on Object-Oriented Information Systems - OOIS '01* (pp. 319–328). Calgary, Canada: Springer.

Fernandes, J., Machado, R., Monteiro, P., & Rodrigues, H. (2006). A Demonstration Case on the Transformation of Software Architectures for Mobile Applications. In *From Model-Driven Design to Resource Management for Distributed Embedded Systems: IFIP TC 10 Working Conference on Distributed and Parallel Embedded Systems - DIPES 2006* (Vol. 225, pp. 235–245). Braga, Portugal: Springer.

Ferreira, N., & Santos, N. (2011). *M207 – ISOFIN Logical Architecture – Part I*.

Ferreira, N., Santos, N., & Soares, P. (2011). *M210 – ISOFIN Financial Domain Applications / Services Specifications*.

Ferreira, N., Santos, N., & Soares, P. (2012). *M207 – ISOFIN Logical Architecture Part II*.

Fowler, M. (2001). Writing The Agile Manifesto. *Martinfowler.com*. Retrieved October 28, 2012, from http://martinfowler.com/articles/agileStory.html

Keith, C. (2008). Why Use Scrum? *Clintonkeith.com*. Retrieved October 17, 2012, from http://www.clintonkeith.com/resources/Why Use Scrum.pdf

Kniberg, H. (2007). Scrum and XP from the Trenches. *Lulu.com*. Retrieved October 27, 2013, from http://tscherning.mono.net/upl/10004/110224ScrumAndXpFromTheTrenches.pdf

Kruchten, P. (1995). Architectural Blueprints - The "4 + 1" View Model of Software Architecture. *Tutorial Proceedings of Tri-Ada*, *12*(November), 42–50.

Li, J., Moe, N., & Dybå, T. (2010). Transition from a plan-driven process to Scrum: a longitudinal case study on software quality. In *Proceedings of the 2010 ACM-IEEE International Simposium on Empirical Software Engineering and Measurement - ESEM' 10* (pp. 1–10). Bolzano, Italy: ACM Press.

Löffler, R., Güldali, B., & Geisen, S. (2010). Towards Model-based Acceptance Testing for Scrum Sprint. *Softwaretechnik-Trends*, *30*(3).

Machado, R., Fernandes, J., Monteiro, P., & Rodrigues, H. (2005). Transformation of UML Models For Service-oriented Software Architectures. In *Procedings of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems - ECBS '05* (pp. 173–182). Greenbelt, USA: IEEE.

Machado, R., Fernandes, J., Monteiro, P., & Rodrigues, H. (2006). Refinement of software architectures by recursive model transformations. In *Proceedings of the International Conference on Product-Focused Software Process Improvement - PROFES '06* (pp. 422–428). Amsterdam, Netherlands: Springer.

Mahnic, V., & Zabkar, N. (2008). Measurement repository for Scrum-based software development process. In *Procedings of the 2nd WSEAS International Conference on Computer Engineering and Applications - CEA'08* (pp. 23–28). Acapulco, Mexico. Retrieved from http://www.wseas.us/e-library/conferences/2008/mexico/cea/1-CEA.pdf

Moe, N., Dingsøyr, T., & Dybå, T. (2008). Understanding Self-organizing Teams in Agile Software Development. In *19th Australian Conference on Software Engineering - ASWEC '08* (pp. 76–85). IEEE.

Moe, N., Dingsøyr, T., & Dybå, T. (2010). A teamwork model for understanding an agile team: A case study of a Scrum project. *Information and Software Technology*, *52*(5), 480–491.

Moore, R., Reff, K., Graham, J., & Hackerson, B. (2007). Scrum at a Fortune 500 Manufacturing Company. In *Proceedings of the Agile Conference - AGILE '07* (pp. 175–180). Washington DC, USA: IEEE.

Paasivaara, M., Durasiewicz, S., & Lassenius, C. (2008). Using Scrum in a Globally Distributed Project: A Case Study. *Software Process: Improvement and Practice*, *13*(6), 527–544.

Paetsch, F., Eberlein, A., & Maurer, F. (2003). Requirements engineering and agile software development. In *Proceedings of 12th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises- WET ICE '03.* (pp. 308–313). Linz, Austria: IEEE.

Rising, L., & Janoff, N. (2000). The Scrum Software Development Process for for Small Teams. *Software, IEEE*, *17*(4), 26–32.

Rockwood, J. (2007). *Choose Your Weapon Wisely - A Handbook for determining the right software development method best suited for your team, client and project*. Carnegie Mellon University.

Rose, T., & Mello, C. (2010). Proposta de Sistemática para Gestão de Projetos baseada na Metodologia Ágil Scrum. In *XXX Encontro Nacional de Engenharia de Produção*. São Carlos, Brasil.

Scharff, C., & Verma, R. (2010). Scrum to support mobile application development projects in a just-in-time learning context. *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering - CHASE '10*, 25–31.

Schatz, B., & Abdelshafi, I. (2005). Primavera gets agile: a successful transition to agile development. *IEEE Software*, *22*(3), 36–42.

Schwaber, K. (1997). SCRUM Development Process. In *Business Object Design and Implementation* (pp. 117–134). London: Springer.

Schwaber, K. (2004). *Agile Project Management with Scrum*. Redmond, USA: Microsoft Press.

Schwaber, K., & Sutherland, J. (2010). What Is Scrum? *Scrumalliance.org*. Retrieved October 11, 2012, from http:/system/resource/file/275/whatIsScrum.pdf

Schwaber, K., & Sutherland, J. (2011). *O Guia do Scrum*. Retrieved from https://www.scrum.org/Scrum-Guides

Scotland, K., & Boutin, A. (2008). Integrating Scrum with the Process Framework at Yahoo! Europe. In *Proceedings of the Agile Conference - AGILE '08* (pp. 191–195). Toronto, Canada: IEEE.

Sutherland, J. (2001). Agile can Scale: Inventing and Reinventing SCRUM in Five Companies. *Cutter IT Journal*, *14*(12), 5–11.

Sutherland, J. (2004). Agile development: Lessons learned from the first scrum. *Cutter Agile Project Management Advisory Service: Executive Update*, *5*(20), 1–4.

Sutherland, J. (2005). Future of Scrum: Parallel Pipelining of Sprints in Complex Projects. In *Proceedings of the Agile Development Conference - ADC '05* (pp. 90–102). Brighton, USA: IEEE.

Sutherland, J., & Altman, I. (2009). Take No Prisoners: How a Venture Capital Group Does Scrum. In *Procedings of the Agile Conference - AGILE '09.* (pp. 350–355). Chicago, USA: IEEE.

Sutherland, J., & Johnson, K. (2008). Scrum and cmmi level 5: The magic potion for code warriors. In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences - HICSS '08* (pp. 466–475). Waikoloa, USA: IEEE.

Sutherland, J., & Schwaber, K. (2010). *The Scrum Papers: Nut, Bolts, and Origins of an Agile Framework*. *SCRUM Training Institute*.

Sutherland, J., Viktorov, A., & Blount, J. (2006). Adaptive Engineering of Large Software Projects with Distributed/Outsourced Teams. In *Procedings of the International Conference on Complex Systems* (pp. 25–30). Boston, USA: IEEE.

Takeuchi, H., & Nonaka, I. (1986). The new new product development game. *Harvard business review*, *64*(1), 137–146.

VersionOne. (2011). *The State Of Agile Development*. Retrieved from http://www.versionone.com/state_of_agile_development_survey/11/

Vlaanderen, K., Jansen, S., Brinkkemper, S., & Jaspers, E. (2011). The agile requirements refinery: Applying SCRUM principles to software product management. *Information and Software Technology*, *53*(1), 58–70.

Woodward, E., Surdek, S., & Ganis, M. (2010). *A Practical Guide to Distributed Scrum* (IBM.). Pearson Education.

Yang, Y., He, M., Li, M., Wang, Q., & Boehm, B. (2008). Phase distribution of software development effort. In *Proceedings of the Second ACM-IEEE International*

Symposium on Empirical Software Engineering and Measurement - ESEM '08 (pp. 61–69). New York, USA: ACM Press.

Ziv, H., & Richardson, D. (1997). The Uncertainty Principle in Software Engineering. In *Procedings of the 19th International Conference on Software Engineering - ICSE '97*. Boston,: IEEE.

Anexo I – Os Doze Princípios do Manifesto Ágil

- Garantir a satisfação do cliente entregando rapidamente e continuamente *software* funcional;
- *Software* funcional é entregue frequentemente (semanas, ao invés de meses);
- *Software* funcional é a principal medida de progresso do projecto;
- Até mesmo mudanças tardias de escopo no projecto são bem-vindas.
- Cooperação constante entre pessoas que entendem do 'negócio' e desenvolvedores;
- Projectos surgem através de indivíduos motivados, e que deve existir uma relação de confiança.
- *Design* do *software* deve prezar pela excelência técnica;
- Simplicidade;
- Rápida adaptação às mudanças;
- Indivíduos e interações mais do que processos e ferramentas;
- *Software* funcional mais do que documentação extensa;
- Colaboração com clientes mais do que negociação de contratos;
- Responder a mudanças mais do que seguir um plano.

Anexo II – *User Stories* do Módulo ISOFIN *App Management*

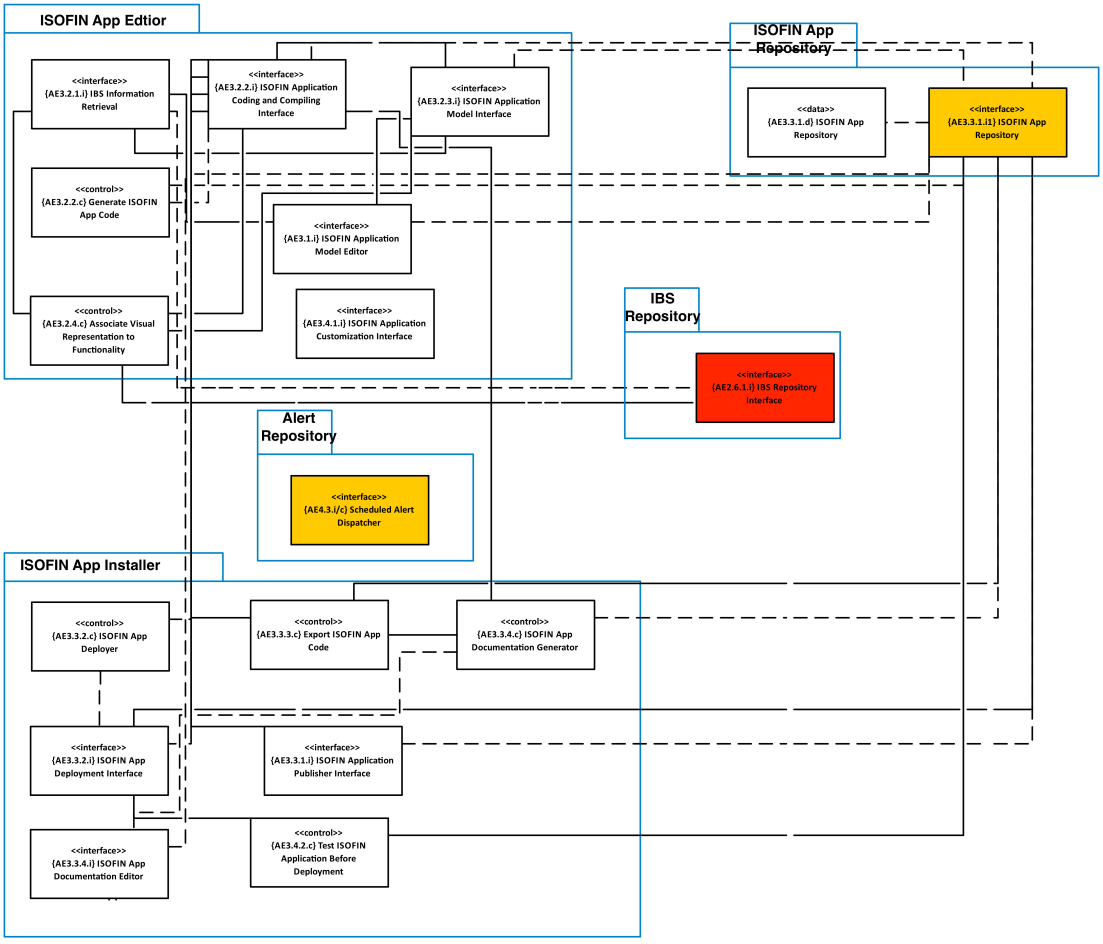

Figura 1 – Diagrama de componentes da aplicação *ISOFIN App Management*

AE #	Nome	As a(n) <actor>	I want/need (to/be able to) <description>	In order to <outcome>
3.2.2.c	Generate ISOFIN App Code	IBS Developer	generate ISOFIN App code	generate ISOFIN App
3.2.4.c	Associate Visual Representation to Functionality	IBS Developer	associate visual representations to functionalities	define interface fields
3.3.2.c	ISOFIN App Deployer	IBS Developer	ISOFIN App deployer	execute ISOFIN App deployment
3.3.3.c	Export ISOFIN App Code	IBS Developer	export ISOFIN App code	deploy ISOFIN Application
3.3.4.c	ISOFIN App Documentation Generator	IBS Developer	ISOFIN App Documentation Generator	automatically generate ISOFIN app documentation
3.4.2.c	Test ISOFIN App Before Deployment	IBS Developer / Business User	test ISOFIN App Before Deployment	render and test ISOFIN App in Pre-Runtime

Tabela 1 – *User Stories* geradas pela aplicação do processo FLAUS aos AEs de controlo da aplicação *iSOFIN App Management*

AE #	Nome	As a(n) <actor>	I want/need (to/be able to) <description>	In order to <outcome>
3.3.1.d	ISOFIN App Repository	IBS Developer	ISOFIN App repository	publish ISOFIN application in catalog

Tabela 2 – *User Stories* geradas pelo processo FLAUS aos AEs de *data* da aplicação *iSOFIN App Management*

AE #	Nome	As a(n) <actor>	I want/need (to/be able to) <description>	In order to <outcome>
3.1.i	ISOFIN App Model Editor	IBS Business Analyst	ISOFIN App model editor interface	model the composition of an ISOFIN application
3.2.1.i	IBS Information Retrieval	IBS Developer	IBS information retrieval interface	provide access to IBS catalogs
3.2.2.i	ISOFIN App Coding and Compiling Interface	IBS Developer	ISOFIN App coding and compiling interface	create ISOFIN app code
3.2.3.i	ISOFIN App Model Interface	IBS Developer	ISOFIN App model interface	create ISOFIN app model and generate ISOFIN app code
3.3.1.i	ISOFIN App Publisher Interface	IBS Developer	ISOFIN App publisher interface	publish ISOFIN application in catalog
3.3.1.i1	ISOFIN App Repository	IBS Developer	ISOFIN App repository interface	access and publish ISOFIN applications in catalogs
3.3.2.i	ISOFIN App Deployment Interface	IBS Developer	ISOFIN App deployment interface	execute ISOFIN App deployment
3.3.4.i	ISOFIN App Documentation Editor	IBS Developer	ISOFIN App documentation interface	provides an interface to allow to get the automatically generated documentation

Tabela 3 – *User Stories* geradas pelo processo FLAUS aos AEs de *interface* da aplicação *iSOFIN App Management*

Anexo III – *User Stories* do Módulo *Alert Management*

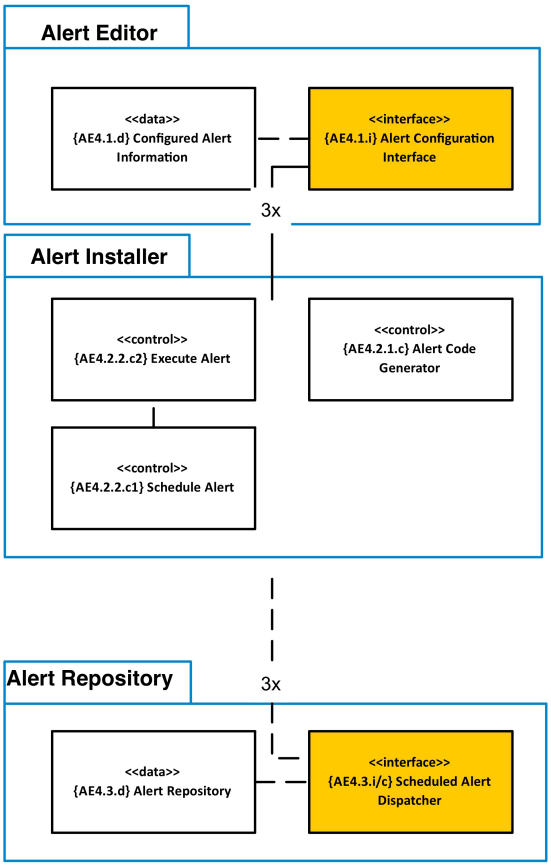

Figura 1 – Diagrama de componentes da aplicação *Alert Management*

AE #	Nome	As a(n) <actor>	I want/need <description>	In order to <outcome>
4.2.1.c	Alert Code Generator	Alert Creator	alert code generator	deploy system alerts
4.2.2.c1	Schedule Alert	Alert Creator	schedule alerts	deploy customized alerts
4.2.2.c2	Execute Alert	Alert Creator	execute alerts	deploy customized alerts
4.3.c	Scheduled Alert Dispatcher	Alert Creator	to have a scheduled alert dispatecher	send alert commands to applications

Tabela 1 – *User Stories* geradas pelo processo FLAUS aos AEs de controlo da aplicação *Alert Management*

AE #	Nome	As a(n) <actor>	I want/need <description>	In order to <outcome>
4.1.d	Configured Alert Information	Alert Creator	access configured alert information	define alerts
4.3.d	Alert Repository	Alert Creator	alert repository	send alert commands

Tabela 2 – *User Stories* geradas pelo processo FLAUS aos AEs de *data* da aplicação *Alert Management*

AE #	Nome	As a(n) <actor>	I want/need <description>	In order to <outcome>
4.1.i	Alert Configuration Interface	Alert Creator	alert configuration interface	configure new alerts

Tabela 3 – *User Stories* geradas pelo processo FLAUS aos AEs de *interface* da aplicação *Alert Management*

Anexo IV – *User Stories* do Módulo *Subscription Management*

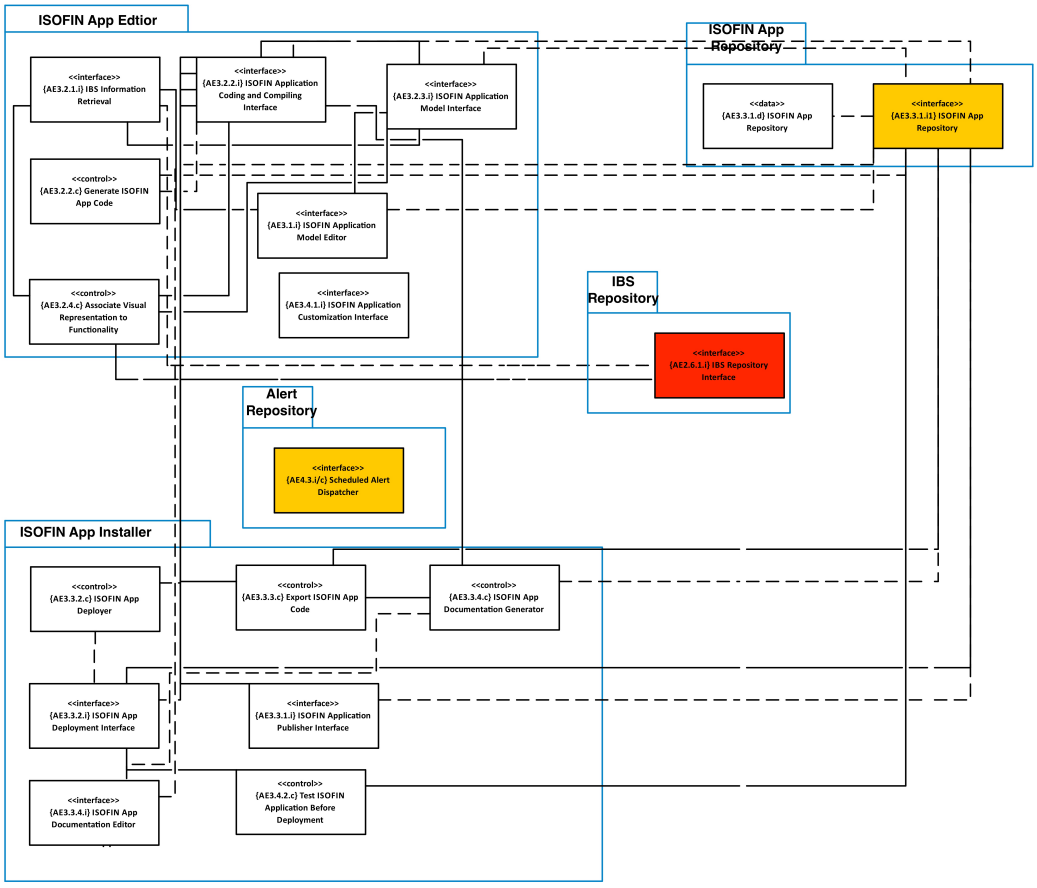

Figura 1 – Diagrama de componentes da aplicação *Subscription Management*

AE #	Nome	As a(n) <actor>	I want/need <description>	In order to <outcome>
1.1.2.c1	Verifiy Subscription data duplicity	IBS Business Analyst	verify subscription data duplicity	verify if subscription requirements information already exists
1.1.2.c	Validate subscription requirement fulfillment	IBS Business Analyst	validate subscription requirement fulfillment	accept candidates
1.3.1.c	Evaluate Supplier subscription	IBS Business Analyst	evaluate supplier subscription	eveluate the impact on ISOFIN Platform of the new subscription request from the supplier
1.3.2.c	Evaluate Customer subscription	IBS Business Analyst	evalute costumer subscription	eveluate the impact on ISOFIN Platform of the new subscription request from the customer
1.3.5.c	Subscription Duplicity Verification	IBS Business Analyst	verify subscription duplicity	verify if information already exists in access subscription repository
1.7.c	Control Subscription Requests	IBS Business Analyst / SBS Supplier / ISOFIN Customer	control subscription requests	communicate subscription request status

Tabela 1 – *User Stories* geradas pelo processo FLAUS aos AEs de controlo da aplicação *Subscription Management*

AE #	Nome	As a(n) <actor>	I want/need <description>	In order to <outcome>
1.3.5.d	ISOFIN Platform Subscription Info Repository	IBS Business Analyst	ISOFIN platform subscription info repository	store subscriptions information
1.4.d	ISOFIN Platform Subscription Assessment Repository	SBS Publisher / SBS Business Analyst / ISOFIN Customer	ISOFIN platform subscription assessment repository	store new subscription requests

Tabela 2 – *User Stories* geradas pelo processo FLAUS aos AEs de *data* da aplicação *Subscription Management*

AE #	Nome	As a(n) <actor>	I want/need <description>	In order to <outcome>
1.1.1.i	Fulfillment Subscription Requirements Interface	ISOFIN Customer / SBS Supplier	fulfillment subscription requirements interface	submit subscription requirements
1.1.2.i	Manual Subscription Validation Interface	IBS Business Analyst	manual subscription validation interfaceverify	verify subscrpition requirement fullfillment
1.3.1.i	Supplier subscription evaluation interface	IBS Business Analyst	supplier subscription evaluation interface	evaluate supplier subscription impact
1.3.2.i	Customer subscription evaluation interface	IBS Business Analyst	customer subscription evaluation interface	evaluate custmer subscription impact
1.3.3.i	Supplier's Subscriptions Requirements Interface	IBS Business Analyst	suppliers subscription requirements interface	set supplier's subscription requirements/parameters
1.3.4.i	Customer's Subscriptions Requirements Interface	IBS Business Analyst	customers subscription requirements interface	set customer's subscription requirements/parameters
1.3.5.i	Subscription Repository Interface	IBS Business Analyst	subscription repository interface	search the subscription repositor for specific subscriptions
1.4.i	Subscription Request Interface	SBS Publisher / SBS Business Analyst / ISOFIN Customer	subscription request interface	request platform subscription
1.5.1.i	Publish SBS Subscsription in Catalog Interface	IBS Business Analyst	SBS subscription in catalog interface	publish SBS subscription information
1.5.2.i	Publish Customer Subscsription in Catalog Interface	IBS Business Analyst	customer subscription in catalog interface	publish customer subscription information
1.7.1.i	Suscription Request Status Interface	IBS Business Analyst / SBS Supplier / ISOFIN Customer	subscription request status interface	communicate subscription request status
1.8.1.i	Manage ISOFIN Suppliers	IBS Business Analyst	manage ISOFIN suppliers	list all ISOFIN SBS information
1.8.2.i	Manage ISOFIN Customers	IBS Business Analyst	manage ISOFIN customers	list all customer informations

Tabela 3 – *User Stories* geradas pelo processo FLAUS aos AEs de *interface* da aplicação *Subscription Management*

Anexo V – *User Stories* do Módulo *Security Management*

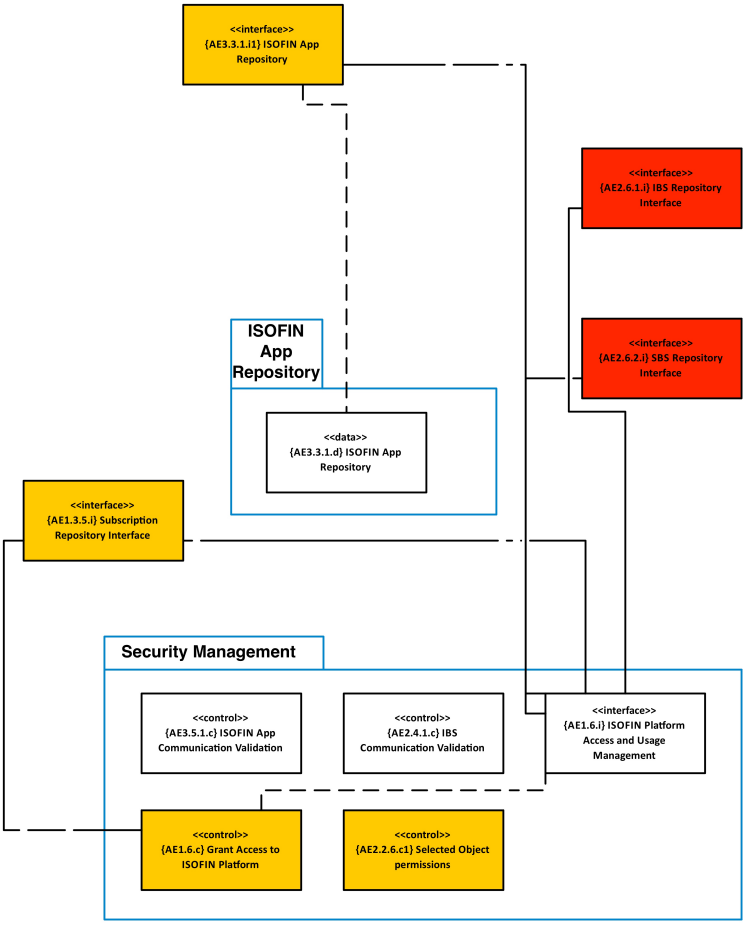

Figura 1 – Diagrama de componentes da aplicação *Security Management*

AE #	Nome	As a(n) <actor>	I want/need <description>	In order to <outcome>
1.6.c	Grant Access to ISOFIN Platform	IBS Business Analyst / SBS Supplier / ISOFIN Customer	to be granted access to ISOFIN platform	to access ISOFIN platform
2.2.6.c1	Selected Object Permissions	IBS Developer	selected object permissions	set and manage permissions and create IBS
2.4.1.c	IBS Communication Validation	SBS Publisher / SBS Developer / ISOFIN Customer	IBS communication validation	send information from the IBS
3.5.1.c	ISOFIN App Communication Validation	ISOFIN Customer	ISOFIN app communication validation	send information from the ISOFIN app

Tabela 1 – *User Stories* geradas pelo processo FLAUS aos AEs de controlo da aplicação *Security Management*

AE #	Nome	As a(n) <actor>	I want/need <description>	In order to <outcome>
3.3.1.d	ISOFIN App Repository	IBS Developer	ISOFIN app repository	publish ISOFIN app in catalog

Tabela 2 – *User Stories* geradas pelo processo FLAUS aos AEs de *data* da aplicação *Security Management*

AE #	Nome	As a(n) <actor>	I want/need <description>	In order to <outcome>
1.6.i	ISOFIN Platform Access and Usage Management	IBS Business Analyst / SBS Supplier / ISOFIN Customer	manage ISOFIN platform access and usage	change and create new permissions for the access and usage of the artifacts registarem in the ISOFIN platform

Tabela 3 – *User Stories* geradas pelo processo FLAUS aos AEs de *interface* da aplicação *Security Management*

Anexo VI – *User Stories* do Módulo *Policies Management*

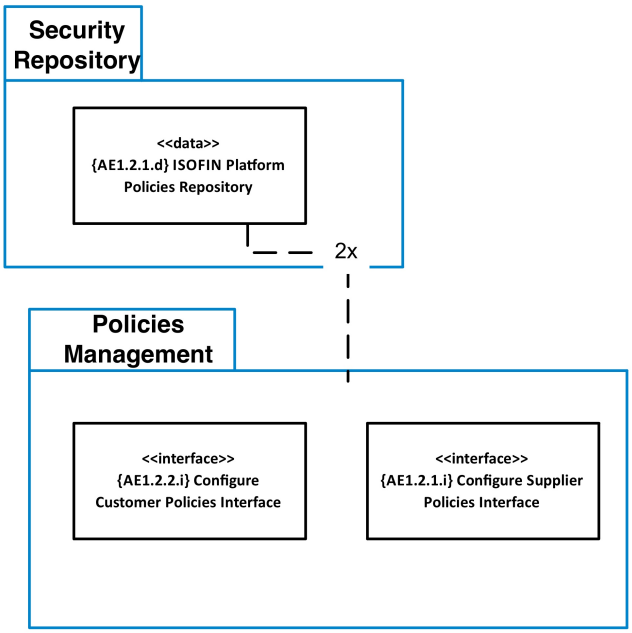

Figura 1 – Diagrama de componentes da aplicação *Policies Management*

AE #	Nome	As a(n) <actor>	I want/need <description>	In order to <outcome>
1.2.1.d	ISOFIN Platform Policies Repository	IBS Business Analyst / SBS Supplier	to access ISOFIN platform policies repository	configure supplier policies

Tabela 1 – *User Stories* geradas pelo processo FLAUS aos AEs de *data* da aplicação *Policies Management*

AE #	Nome	As a(n) <actor>	I want/need <description>	In order to <outcome>
1.2.1.i	Configure Supplier Policies Interface	IBS Business Analyst / SBS Supplier	supplier policies configuration interface	configure supplier policies
1.2.2.i	Configure Customer Policies Interface	IBS Business Analyst / ISOFIN Customer	customer policies configuration interface	configure customer policies

Tabela 2 – *User Stories* geradas pelo processo FLAUS aos AEs de *interface* da aplicação *Policies Management*

Anexo VII – *User Stories* do Módulo *Logs Management*

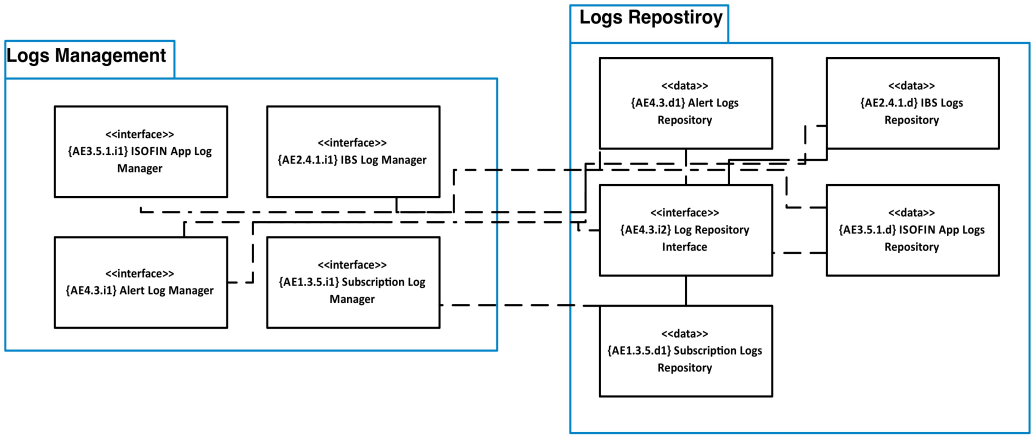

Figura 1 – Diagrama de componentes da aplicação *Logs Management*

AE #	Nome	As a(n) <actor>	I want/need <description>	In order to <outcome>
1.3.5.d1	Subscriptions Logs Repository	IBS Business Analyst	subscriptions logs repository	store reportes from the request and menagens of the subscriptions
2.4.1.d	IBS Logs Repository	SBS Developer / SBS Publisher / ISOFIN Customer	IBS logs repository	store information abou IBS usage
3.5.1.d	ISOFIN App Logs Repository	ISOFIN Customer	ISOFIN app logs repository	store information abou ISOFIN app usage
4.3.d1	Alert Logs Repository		alert logs repository	store the usage of the scheduled alert despache

Tabela 1 – *User Stories* geradas pelo processo FLAUS aos AEs de *data* da aplicação *Logs Management*

AE #	Nome	As a(n) <actor>	I want/need <description>	In order to <outcome>
1.3.5.i1	Subscription Log Manager	IBS Business Analyst	subscription log manager interface	access subscription log repository
2.4.1.i1	IBS Log Manager	SBS Developer / SBS Publisher / ISOFIN Customer	IBS log manager interface	access IBS logs repository
3.5.1.i1	ISOFIN App Log Manager	ISOFIN Customer	ISOFIN app log manager interface	access ISOFIN app log repository
4.3.i1	Alert Log Manager	Alert Creator	alert log manager interface	access alert log repository
4.3.i2	Log Repository Interface	Alert Creator	log repository interface	access and manage all the logs repository

Tabela 2 – *User Stories* geradas pelo processo FLAUS aos AEs de *interface* da aplicação *Logs Management*